Advances in COMPUTERS

VOLUME 29

Edited by

MARSHALL C. YOVITS

Advances

in COMPUTERS

VOLUME 29

Contributors to This Volume

JOHN M. CARROLL
ROBERT W. CLOUGH
RICHARD W. JUDY
MING T. LIU
JONATHAN K. MILLEN
MONROE NEWBORN

Advances in
COMPUTERS

EDITED BY

MARSHALL C. YOVITS

Purdue School of Science
Indiana University—Purdue University of Indianapolis
Indianapolis, Indiana

VOLUME 29

ACADEMIC PRESS, INC.
Harcourt Brace Jovanovich, Publishers
Boston San Diego New York
Berkeley London Sydney
Tokyo Toronto

COPYRIGHT © 1989 BY ACADEMIC PRESS, INC.

ALL RIGHTS RESERVED.
NO PART OF THIS PUBLICATION MAY BE REPRODUCED OR TRANSMITTED IN ANY FORM OR BY ANY MEANS, ELECTRONIC OR MECHANICAL, INCLUDING PHOTOCOPY, RECORDING, OR ANY INFORMATION STORAGE AND RETRIEVAL SYSTEM, WITHOUT PERMISSION IN WRITING FROM THE PUBLISHER.

ACADEMIC PRESS, INC.
1250 Sixth Avenue, San Diego, CA 92101

United Kingdom Edition published by
ACADEMIC PRESS INC. (LONDON) LTD.
24-28 Oval Road, London NW1 7DX

LIBRARY OF CONGRESS CATALOG CARD NUMBER: 59-15761

ISBN 0-12-012129-8

PRINTED IN THE UNITED STATES OF AMERICA

89 90 91 92 9 8 7 6 5 4 3 2 1

Contents

Computer Chess: Ten Years of Significant Progress

Monroe Newborn

Soviet Computing in the 1980s

Richard W. Judy and Robert W. Clough

Contributors

Numbers in parentheses refer to the pages on which the authors' contributions begin.

John M. Carroll (47), *User Interface Institute, IBM T. J. Watson Research Center, Box 704, Yorktown Heights, New York 10598*

Robert W. Clough (251), *Hudson Institute, Herman Kahn Center, 5395 Emerson Way, PO Box 26–919, Indianapolis, Indiana 46226*

Richard W. Judy (251), *Hudson Institute, Herman Kahn Center, 5395 Emerson Way, PO Box 26–919, Indianapolis, Indiana 46226*

Ming T. Liu (79), *Department of Computer and Information Science, The Ohio State University, 2036 Neil Avenue, Columbus, Ohio 43210–1277*

Jonathan K. Millen (1), *The MITRE Corporation, Burlington Road, Bedford, Massachusetts 01730*

Monroe Newborn (197), *School of Computer Science, McGill University, Montreal, Quebec, Canada H3A 2A7*

This Page Intentionally Left Blank

Preface

The serial, *Advances in Computers*, provides a medium for the in depth presentation of subjects of both current and long-range interest to the computer and information community. Within this framework, contributions for appropriate articles have been solicited from widely recognized experts in their fields. The time scale of the invitation is such that it permits a relatively leisurely perspective. Furthermore, the permitted length of the contributions is greater than many other publications. Thus, topics are treated both in depth and breadth.

The serial began in 1960 and now continues with Volume 29. These books have played an important role over the years in the development of the computer and information fields. As these fields have continued to expand—both in research and resulting applications as well as in their significance—so does the importance of the *Advances* series. As a consequence, it was decided that Academic Press would this year publish two volumes, 28 and 29; Volume 28 was published earlier this year.

Included in Volume 29 are chapters on computer security, human-computer interaction, protocol engineering, computer chess, and Soviet computing.

In the first chapter, Dr. Millen considers the very important current issue of computer security. He points out that multilevel security implies the assignment of labels, such as classification levels, to data and users in order to control access. Classification labels bring to mind military applications, with labels such as "Confidential" and "Top Secret," but other sets of labels are useful in a commercial environment. In considering some multilevel access-control models, Millen focuses on a few influential ideas rather than on secure systems in general. Certain models are examined in detail because of the ideas they express and the questions they raise. He explains that the developments in information-flow modelling are exciting because they are still evolving in a clear direction. The underlying notion of information flow as an inference about the possible values of sensitive data sources had led to the important noninterference concept in deterministic machines.

John Carroll considers the area of human–computer interaction. He likens the recent evolution of computer technology to that of a "race" between function and usability. The frontier of usability has been pressed onward by the development of new applications and interface technologies. The race between function and usability, he states, has made the area of human–computer interaction a very high-profile research area within computer science and within the computer industry. It is difficult to develop science and technology relating to usability rapidly enough, but it is critical to do so. Human–computer interaction has often been described as an interdisciplinary

research area, but only now are the full interdisciplinary possibilities emerging, with psychologists participating fully and in a variety of roles in the evolution of computer technology.

Professor Liu in the third chapter is concerned with computer-communication protocols. These are sets of rules permitting an orderly exchange among physically separated computers. The discipline in this area is now called *protocol engineering* and is currently receiving increased attention. Dr. Liu shows that a protocol engineering system allows the protocol designer to express the protocol formally, test its specifications for correctness (validation and verification), obtain some early indication of how it performs, compile major parts of the implementation directly from the formal specifications, and, finally, test the resultant implementation to assure that it conforms to specifications (implementation verification or conformance testing).

Professor Newborn contributed a chapter to Volume 18 of *Advances in Computers* 10 years ago, which surveyed developments in computer chess in the middle and late 1970s that raised the playing strength of chess programs to just over the 2000 level, the United States Chess Federation Expert rating. Now chess programs have improved at least another 500 rating points and are playing almost at Grandmaster level. In this chapter, Newborn describes the technical developments that led to this remarkably strong level of play. He goes on to indicate that while the last decade has seen programs progress from playing at the Expert level to almost that of the Grandmasters, the coming decade should be even more exciting. It is quite likely that before the year 2000, a computer will defeat the human world champion.

Dr. Richard Judy and Robert Clough state that Soviet computing in the 1980s has been a very interesting scene. This was the decade when the nation's top political leadership finally recognized the central role of computers and other information technologies in military, economic, and social development. This recognition however came very late in the game, not before the Soviet Union's international competitors attained a huge, perhaps insurmountable, lead in the technologies and their applications. Compared with Western and Japanese progress in developing and using information technologies of all kinds, the Soviet Union has continued to lose ground rapidly in the 1980s. Judy and Clough point out that there is mixed news for the Soviet computer user of the late 1980s. Available hardware and software continue to fall further behind what their Western counterparts are using at every level, from supercomputers to microcomputers. This however is tempered by the fact that the scientific and political leadership now openly recognizes the problem and vows to resolve it.

It is my great pleasure to thank the contributors to this volume. They have given extensively of their time and effort to make this book an important and timely contribution to their profession. Despite the many calls upon their time,

they recognized the necessity of writing substantial review and tutorial articles. It has required considerable effort on their part, and their cooperation and assistance is greatly appreciated. Because of their efforts, this volume achieves a high level of excellence, that should be of great value for many years to come. It has been a pleasant and rewarding experience for me to edit this volume and to work with these authors.

MARSHALL C. YOVITS

This Page Intentionally Left Blank

Models of Multilevel Computer Security

JONATHAN K. MILLEN

The MITRE Corporation
Burlington Road
Bedford, Massachusetts

1. Introduction

1.1 Nondiscretionary Security Policy

Multilevel security implies the assignment of labels, such as classification levels, to data and users, for the purpose of controlling access. Classification labels bring to mind military applications, with labels such as "Confidential" and "Top Secret," but other sets of labels may be useful in a commercial

ADVANCES IN COMPUTERS, VOL. 29

Copyright © 1989 by Academic Press, Inc.
All rights of reproduction in any form reserved.
ISBN 0-12-012129-8

environment (Lipner, 1982). In practice, label-based controls are supplemented by additional access restrictions. Some of the special policies appropriate for commercial applications are discussed by Clark and Wilson (1987).

Access-control policies on label assignments are termed "nondiscretionary" or, synonymously, "mandatory." Policies in which ordinary users can decide whether or not to grant or transfer access privileges to other users, for acess to certain data under their control, are referred to as "discretionary." In systems with a discretionary policy, it can be difficult to determine the extent to which access rights propagate. This general problem is the *safety problem*, and it has been shown to be recursively undecidable in a sufficiently broad context (Harrison, 1985; Harrison, *et al.*, 1976).

Nondiscretionary access-control models are interesting primarily because of their role in a process of implementation that has been reasonably successful, rather than because of any deep mathematical results. One clear and simple reason for implementing a nondiscretionary policy is to foil "Trojan horse" programs. Such programs cannot reassign labels, and hence cannot affect label-based access restrictions. By contrast, in a purely discretionary system, they might reassign access permissions, or move information, without the knowledge of, and against the intent of, the human user on whose behalf the program is supposed to be executing.

Some multilevel access-control models will be surveyed here. There have been other surveys, such as those by Landwehr (1981) and Millen and Cerniglia (1984). This survey includes more recent models, and also differs from previous ones by presenting a few models in greater depth. We wish to focus on a few influential ideas rather than models or secure systems, and there will be no attempt at broad coverage of either old or new models. Certain models will be examined in detail because of the ideas they express and the questions they raise.

The first access-control models were for operating systems, and modelled the policy by which an operating system grants requests by processes for access to controllable segments of main memory. We shall look at the design decisions behind these models, and their intended application to secure computer system development. Some new ideas arise in database system models, which impose additional structure on data objects, raising questions about how to assign labels. There will be a brief discussion of network models.

It has been known, at least since an article by Lampson (1973), that unauthorized disclosure of data is possible even in a system where access controls are perfectly enforced, and even if they are nondiscretionary. Computer systems may have *leakage channels* or *covert channels* by which a process accessing sensitive data may communicate it to a user who is not supposed to have access to that data. We shall conclude with a few models of information flow that are deep enough to explain this phenomenon, and which

have given rise to techniques for detecting information flow in violation of label-based policies. These models are not access-control models.

1.2 Reference Monitors

1.2.1 Subjects, Objects, and Access

In its simplest form, an access-control model has *subjects*, or active entities, that can exercise various modes of *access* on *objects*, or repositories of information. Several modes of access may be recognized. Access is a directed relation: subjects have access to objects. If one desires to model some form of access by one subject to another, subjects can be assumed to be a special kind of objects. The model can be thought of as a state-transition machine whose current state is an *access matrix* showing, for each subject and object, what set of access modes the subject currently has for that object. An abstract machine of this kind is called a *reference monitor*. These basic ideas come from Lampson (1971) and Graham and Denning (1972); the term "reference monitor" arose in a U.S. Air Force planning study (Anderson, 1972).

Access modes, in a multilevel context, have implications for information flow. In particular, each access mode must be interpreted as representing a *read*, a *write*, both, or neither, in addition to whatever other significance it might have. If a subject has some form of read access to an object, information can flow from the object to the subject. If a subject has some form of write access to an object, information can flow from the subject to the object.

1.2.2 Subject Memory

Subjects are viewed as agents for transmitting information. If a subject has simultaneous read access to one object and write access to another, information flows from the first object through the subject to the second object. What if a subject has read access to an object temporarily, but releases that access before obtaining write access to another object? Does information flow from the first object to the second? This is equivalent to asking whether subjects have memory. It is usually assumed that they do.

Some modellers prefer to say that subjects have no memory themselves, but each subject is associated with a private object to which it has read and write access. For example, if one thinks of a process in an operating system as a subject, its private object consists of its processor context—i.e., its registers. The problem with this approach is that it is rarely followed through conscientiously. No one ever bothers to specify axiomatically in their models that such private objects always exist with read/write access. And when they try to show that software specifications or high-order language code satisfies

the model, they have trouble because certain private objects are not visible in the implementation.

Certain subjects and objects are part of the external interface of a system, in the sense that they may act as conduits for information entering or leaving it. It is assumed that any such information flow is consistent with the labels on the subjects or objects. Some models make the external interface activities explicit, others do not.

1.2.3 Access Modes and Transactions

Models of higher-level services such as database management systems or message systems frequently emphasize the notion of a "transaction" as the way a user interacts with the system. In a transaction-oriented model, all information flow occurs during transactions. The same transaction may cause read and write accesses to several objects by the requesting subject. In the time between transactions, a subject might perform some local processing on its private memory, but it cannot read or modify other objects.

In this kind of model, the transactions themselves are the modes of access. The access matrix lists, for each subject and object, what transactions the user may employ upon that object.

1.3 Label-Based Policy

1.3.1 The Dominance Relation on Labels

The same set of labels is used for both subjects and objects. On an object, a label represents some measure of the sensitivity of, or special restrictions on, the data in the object. On a subject, the label represents the clearance or privileges of the subject, as well as the sensitivity of the data in its memory. Labels are ordered, in that one can tell, at least for some pairs of labels, when one represents greater data sensitivity than the other. In the tradition of Bell and LaPadula (1975), this ordering—actually a partial ordering, as we shall see in a moment—is often called *dominance*. Dominance will be symbolized in this discussion with the inequality symbol "$\geq$," and the reverse relation "dominated by" with "$\leq$."

1.3.2 Information Flow Policy

Labels are used in models of multilevel security to constrain access. The access restrictions are intended to enforce a higher-level, informal, information-flow policy: *information flow from one entity to another is possible only when the destination carries a label dominating that of the source.* Given

some natural, intuitive properties of information flow, we can show that dominance should be a partial ordering, if this information-flow policy is to be satisfied; these arguments were given by Denning (1976).

First, information flow is trivially possible from an object to itself. Hence, if x is the label of the object, we must have $x \leq x$. So dominance is reflexive.

Second, if information flows from an object a to a subject b, and then from b to an object c, information may have flowed as a result from a to c. Now, suppose the labels on a, b, and c are x, y, and z, respectively. If $x \leq y$ and $y \leq z$, the policy requires $x \leq z$. So dominance is transitive.

Third, suppose there is a subject a and an object b with labels x and y, respectively, such that $x \leq y$ and $y \leq x$. There is nothing that will force us to conclude that $x = y$. We can only say, at this point, that dominance is a pre-ordering, as is done by Walter *et al.* (1974a). However, since information may flow in both directions between a and b, the policy permits a and b to swap information until they contain exactly the same data. There is then no reason to distinguish the labels x and y. So we may as well assume that dominance is antisymmetric also, making it a partial ordering.

A total ordering is not necessary, or always desirable; we do not need to assume that all pairs of labels are comparable. One common and useful system of labels arises from marking each subject and object with one or more categories, e.g., financial, administrative, NATO. A single label is a set of categories. A subject can read an object only if it is cleared for all the categories in the object's label, so the dominance relation in this case is just set inclusion, which is not a total ordering.

It can be convenient also to assume that a least upper bound operator exists for the dominance partial ordering. Suppose one wishes to create a subject to read information from two objects labelled x and y. It is desirable to label the new subject with the least upper bound of x and y. For, suppose there is another object whose label dominates both x and y. Then the new subject can be allowed to write into it.

Denning (1976) points out that when a least upper bound operator exists, and when the set of labels is finite and possesses a universal lower bound (representing non-sensitive, unrestricted information), a greatest lower bound also exists. (The greatest lower bound of x and y is the least upper bound of all common lower bounds of x and y.) Under these circumstances the dominance relation forms a lattice.

1.3.3 The *-Property

In an access-control model, the information-flow policy stays behind the scenes as a motivation for an explicit access-control policy. The access-control policy limits the modes of access permitted between subjects and objects, on

the basis of their labels. Two access-control restrictions are assumed:

- A subject is permitted read access to an object only if its label dominates the label on the object.
- A subject is permitted write access to an object only if its label is dominated by the label on the object.

The conjunction of these two statements is often called the **-property*, after a similar property stated in the Bell-LaPadula model (Bell and LaPadula, 1975). All multilevel access-control models have some form of it. The reader is warned, however, that "*-property" is not a well-defined term. Even Bell and LaPadula have given different versions of it (LaPadula and Bell, 1973; Bell, 1973).

It is obvious, given the kind of assumptions we have made about information flow, that the *-property implies the information-flow policy. Information flow was assumed to occur only as a result of accesses, and the *-property says it can only flow "uphill" with respect to the labels.

2. Implementing Models

2.1 The Successive Refinement Approach

A model is just the first step in a secure system development. One paradigm for using models in implementing secure systems was suggested by the Air-Force planning study (Anderson, 1972), and was refined through a series of projects, initially sponsored by the Air Force and later supported more generally by the U.S. Department of Defense. This line of development led to the publication of the "Orange Book," a standard for evaluating the security of computer systems, by the National Computer Security Center (1985).

The idea proposed in the Preface of the Anderson report was to "... start with a statement of an ideal system, a model, and to refine and move the statement through various levels of design into the mechanisms that implement the model system." At this time, David Parnas had already described a technique for precise specification of software modules (Parnas, 1972). Some work at MITRE (Burke, 1974) put the two together, and recommended the four-stage approach illustrated in Fig. 1. The specifications in stage 2 were called "formal" specifications to distinguish them from imprecise natural-language specifications, and to emphasize the possibility that one could construct mathematical proofs relating the specifications to the model in stage 1, and perhaps also to the high-order-language source code in stage 3.

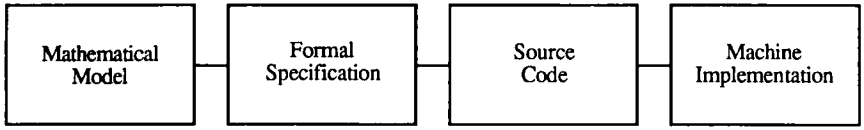


FIG. 1. The four-stage approach.

Carrying out the four-stage approach rigorously was then, and is still now, beyond the state of the art. Showing that the source code satisfies the formal specifications is essentially a proof of program correctness. In 1977, Stanford Research Institute had made considerable progress with a methodology for proving that a hierarchically structured operating system satisfied formal specifications in the spirit of the ones Parnas had suggested (Neumann *et al.*, 1977). Various efforts have been made along these lines, and we shall not attempt to survey them here. None have been fully satisfactory, primarily because program correctness, in general, is a formidable goal that has not been reduced to practice. And even if one could prove that the source code is correct, there is still the question of showing that it has been compiled properly and runs correctly on the target computer.

Despite the failure to apply mathematical rigor to all aspects of the implementation of a secure computer system, techniques traceable to the four-stage approach have led to the development of a few systems that are believed to be much safer than any that had previously been built. Much of the credit for their success goes to improvements in hardware, but without a simple, elegant policy to support, the hardware features might well have been ineffective.

The key ideas in the development of secure multilevel systems have been these:

- A simple, uniformly applied hardware mechanism for protection of memory.
- A small operating system "kernel" that uses the hardware mechanism to protect itself and to control all memory accesses in accordance with a security policy.
- A simple, nondiscretionary policy for the kernel to support.

There are, of course many details that must be thought out and implemented carefully, from interrupt handling and I/O management to user authentication. The reason this approach has a chance of succeeding is that the kernel is designed from the beginning with a full understanding of the hardware mechanism underneath and the policy to be supported.

The *Department of Defense Trusted Computer System Evaluation Criteria*, known as the "Orange Book" (National Computer Security Center, 1985), is a

requirements document used for rating computer systems with respect to their ability to protect classified or other sensitive information. The rating process and its application to Defense systems is too complex to be discussed here. The significance of the Orange Book, for us, is that it embodies much of the experience gained from DoD-sponsored secure computer system development along the lines summarized above.

2.2 Formal Top-Level Specifications

The highest Orange Book rating, "A1," has requirements for implementing multilevel security with the greatest degree of assurance that is considered "reduced to practice." It requires not only a model of a mandatory access-control policy, but also a "formal top-level specification" and evidence that the specification is consistent with the model. Although the Orange Book requires only a "mixture of formal and informal methods" to establish the correspondence, most efforts to meet A1 requirements have used formal methods or verification tools. The Orange book A1 requirements also call for a mapping between the formal specification and the security-critical source code; this requirement is typically satisfied with systematic but informal methods. The first computer system named as satisfying A1 requirements was the Honeywell SCOMP (Fraim, 1983), and it has defeated attempts at penetration.

The intended role of a model, then, is to state a security policy to be supported by system software and hardware, and consequently to serve as a statement of requirements to be satisfied by the next stage of the implementation, the formal specification. In this way, logical errors in the design of the system might be caught earlier, and more easily, than otherwise.

The use of nonprocedural formal specifications has been helpful in making this process practical. It was Parnas (1972) who suggested writing specifications nonprocedurally. A nonprocedural specification says what the result of a function procedure call is, without saying how it is accomplished. It uses logical operators, even quantifiers, but does not employ programming constructs for flow of control, such as sequencing or looping. Conditional statements are used, but they are viewed as logical connectives.

Some languages for formal specification have been developed to the point where software tools exist for parsing specifications in the language and for proving properties about them. Four specification and verification environments of this kind were described by Cheheyl *et al.* (1981), showing how the tools were applied to a small example of a secure system.

2.3 A Flaw Discovered

Here is an example of how a security flaw was discovered; it actually happened during the development of a certain secure system, according to

Guttman (1987). The operating system has a command to create a (logically) new memory segment; the command is called "create_segment." The create_segment command may give the calling process access to the new segment. The process indicates the desired security level label and access mode via arguments to the create_segment call.

The problem arose because a process was allowed to create a segment at a level equal to or higher than its own level. Naturally, when a process creates a segment at a strictly higher level, it should not be able to obtain read access to it. Otherwise, it would be able to read higher-level data written into the segment by any higher-level process. The create_segment command was supposed to check for that, and raise an error condition if the calling process requested read access to a new segment at a strictly higher level. But there was a mistake in the command as specified, and the source code implemented the mistake.

This problem was found while attempting to show that the formal specification was consistent with the model. We can see how this was done by looking at an excerpt from the formal specification. The part of the specification shown below shows the error test within the create_segment command; it states a condition upon which an "invalid_request" error is reported.

```
create_segment (map, wire, access, seg_access, pl):
    if map and
    ((wire and not (access = {'write'}) and not Lteq(seg_access, pl))
    or ...)
    then return 'invalid_request'
    else ...
```

The condition is complicated because the command has various options passed as parameters. Two of these are *map*, a boolean indicating whether the calling process will be given access to the new segment; and *wire*, a boolean indicating whether the new segment is to be locked in main memory. Other arguments are: *access*, the mode of access requested; *seg_access*, the security level specified for the new segment; and *pl*, the security level of the calling process. The condition compares seg_access with pl using Lteq, the "less than or equal to" relation.

The problem is that the security level comparison is made only if "wire" is requested. So if "wire" is not requested, the calling process can get read access to a new segment at a higher level. The condition is hard enough to read so that the specifier and implementor thought they were doing the right thing. The verifier, however, tried to prove a property required by the model:

```
if (map and (access = {'read'} or access = {'read', 'write'}))
then Lteq(seg_access, pl).
```

This property is part of the *-property as stated in the previous section, mapped down into the terminology of the specification, and interpreted for the access state resulting from the create_segment command. It should be provable under the hypothesis that the invalid_request error did not occur (if the error did occur, then create_segment returns immediately with the error message without creating the new segment). But it was not provable, and thus the problem was discovered.

3. Model-to-Specification Correspondence

3.1 Introduction

Mapping model properties down into specification terminology, and then proving them, is one way to show that a specification is consistent with a model. This activity is something like doing program correctness proofs, but there are some important differences:

- The properties to be proved are derived in a uniform way from an abstract model.
- The target of the verification is not a procedure written in source code, but rather a formal specification.
- It is well within the state of the art to construct rigorous proofs for specifications of real systems.

Let us examine what it means, in general, to say that a specification is "consistent with" a model. We shall begin by characterizing models and specifications abstractly, and end with a description of what proving the correspondence implies in a practical sense. The reader is cautioned that this characterization does not apply to all possible models of computer security or forms of system specification; it is appropriate only for the implementation paradigm summarized above. In particular, it is meant for multilevel security models of the sort surveyed in this article, and specifications written in a certain style.

3.2 Abstract Definition of a Secure System

Both models and specifications describe state-transition automata. A *state-transition automaton* (or, simply, a *machine*) includes a set of states Q, a set of inputs X, and a transition function δ from $Q \times X$ to Q. It also has a set of

outputs, Y, which may be associated either with transitions or states, and a specified initial state.

Security models have additional structure. A *reference monitor* can be characterized as a machine that associates an access matrix with each state. Sets of subjects, S, objects, O, and access modes, M, are the additional elementary sets that occur in the definition of a reference monitor. Formally, an access matrix is a function from (subject, object) pairs to sets of access modes; or, it could be a relation containing (subject, object, access-mode) triples.

The access matrix changes from state to state. We could simply define the set of states as the set of access matrices, but usually a state has other components as well. Rather than think of the state as a complicated structure, it may be easier to think of the state set as a collection of state names, identifiers, or indices. Components of a state, such an access matrix, are found using functions defined on the state set. Thus, an access matrix function might be of the form $\alpha: Q \to \mathcal{P}(M)^{S \times O}$ (where the exponent notation A^B represents the set of functions on B into A, and $\mathcal{P}(A)$ is the set of subsets of A).

A *multilevel access-control (MAC) system* is a reference monitor with a partially ordered set L of labels and a function λ associating subjects and objects with their labels. In general, the label assignment is a component of the state, so that λ is of the form $\lambda: Q \to L^{S \cup O}$. It defeats the purpose of a label assignment if labels can change arbitrarily, so most MAC system models restrict such changes. In some models, label assignment is fixed for all states; in others, labels may change only in response to inputs in a distinguished set associated with a trusted source.

Most MAC system models obey some form of the *-property. There might be a distinguished subset of "trusted" subjects, however, whose accesses are permitted to be in violation of the *-property.

At this point, the world of security models diverges. The next step in the progression of models of secure machines is to describe a security policy that is more or less specific to an intended application. We must split off in different directions to reach various application models: models of operating systems, database systems, networks, etc. And when we descend further to formal top-level specifications, the family of systems being described is narrowed even more tightly, to the point where they receive brand names such as "Multics" and "SCOMP."

The idea behind the successive refinement of the concept of a secure system remains the same as we descend through the levels of refinement. Just as each MAC system is a reference monitor, each instance of a given application model is a MAC system. And a formal top-level specification refines an application model similarly.

3.3 Models as Logical Systems

3.3.1 Axioms and Valid Interpretations

When looking at application models and formal specifications, it is helpful to become more conscious of a model as a logical system, with symbols for constants, variables, sets, and relations, and axioms constraining the various functions and relations that are mentioned. The fact that the "dominates" relation on labels is a partial ordering, for example, is expressed with three axioms. The *-property is an axiom constraining the component extractor function associating access matrices with states.

Models may also have axioms restricting state transitions. When the function assigning labels to objects can change from state to state, one might have an axiom stating that objects cannot be downgraded—an object label in the next state dominates the object label in the current state.

A *valid interpretation* of a model is a relational structure (e.g., a machine) together with a mapping of the sets, functions, and relations to the symbols in the model, in such a way that the axioms are true. A mapping of a collection of sets and relations to the MAC system model is accomplished by identifying which set is the set of subjects, which relation is the partial order on labels, etc. Once this correspondence is defined, each model axiom is metamorphosed into a statement about the relations defining the machine. This view of what it means for a machine to be an instance of a model was applied in an early security modelling context by Walter *et al.* (1974b).

It is worth noting that a mapping of sets and relations includes a mapping of individual constants, such as the access modes "read" and "write." This is because constants are viewed as functions of no arguments (and functions are single-valued relations).

3.3.2 Concrete Models and Transition Rules

Some models have axioms of a specific kind called *transition rules*. These are associated with system commands, somewhat like the HRU model (Harrison *et al.*, 1976). Typical operating system commands are "create object," "get access," etc. The Bell-LaPadula model has a set of rules motivated by a design for a secure Multics kernel (Bell and LaPadula, 1975).

The idea behind a transition rule is that an input to the machine has a particular form, namely a command name followed by a list of parameters, e.g., "get_access(subject, access, object)." Actual values must be substituted for the formal parameters to obtain a particular input. A single rule covers all transitions possible with inputs having a given command name.

Let us refer to a model without transition rules as an *abstract* model, and one that includes transition rules a *concrete* model.

The usual convention, established by the Bell-LaPadula model, is that transition rules are not independent of the other axioms. To describe the role of axioms in a concrete model, we shall call attention to two particular sorts of axioms:

- *State invariants*, which must be satisfied by each individual state (i.e., an axiom of the form $\forall q \in Q, P(q)$, where P does not mention any state other than q.
- *Transition* axioms, which mention the transition function.

Transition rules are themselves transition axioms, but they are supposed to furnish a complete, self-contained specification of what transitions are possible. This gives us the first consistency property for concrete models:

(C1) The transition rules, taken together, must imply all other transition axioms.

There is a second consistency property:

(C2) The transition rules preserve all state invariants.

That is, if a (current state, next state) pair is consistent with a transition rule, then the truth of the state invariants for the next state must be provable from the transition rule and the truth of the state invariants for the current state.

The fact that transition rules preserve state invariants is not enough by itself to ensure that all states satisfy the invariants. We need to assume that

- The initial state satisfies the state invariants.
- All states are reachable from the initial state.

These are, in effect, new axioms added implicitly to any concrete model.

The two requirements, (C1) and (C2), in the presence of the new implicit axioms, ensure that the transition rules enforce the state invariants and transition axioms. For this sort of model, these requirements may be what the Orange Book is referring to in Section 3.2.3.2.2, in its requirement for a model that "... is proven consistent with its axioms" (National Computer Security Center, 1985).

3.3.3 *Transition Rule Example*

Here is a partial illustration showing the consistency of a transition rule with a state invariant. Consider this typical transition rule for a read-access

request, from an imaginary concrete model:

get_read_access(cur_proc, segment_id) [cur_state, next_state]:
((label(segment_id) ≤ label(cur_proc))
∧ (access_matrix(next_state, cur_proc, segment_id)
= {'read'} ∪ access_matrix(cur_state, cur_proc, segment_id))
∨ access_matrix(next_state, cur_proc, segment_id)
= access_matrix(cur_state, cur_proc, segment_id))
∧ (($p \neq$ cur_proc ∨ $i \neq$ segment_id) →
access_matrix(next_state, p, i) = access_matrix(cur_state, p, i))

This rule is suppose to show the relation between the access_matrix components of two states related by a transition caused by a get_read_access input. It permits the subject cur_proc to gain read access to the object segment_id, provided that an appropriate test on their labels is satisfied. The test is, of course, motivated by the *-property.

The state invariant that happens to be an axiom of this concrete model is this one, intended to be an interpretation of the *-property:

('read' ∈ access_matrix(q, p, i) → label(i) ≤ label(p))
∧ ('write' ∈ access_matrix(q, p, i) → label(p) ≤ label(i)).

To show that this property is preserved by get_read_access, we must show that it is true with q = next_state whenever it is true with q' = cur_state (the induction hypothesis), assuming that the input was a get_access command with parameters segment_id = i and cur_proc = p.

Looking at the rule, we see that there is only one case where the access_matrix changes from cur_state to next_state. In this case, we have

label(i) ≤ label(p),

and then the rule says that

access_matrix(q, p, i) = {'read'} ∪ access_matrix(q', p, i).

The first conjunct of the *-property is clearly satisfied, and the second conjunct is true by the induction hypothesis.

3.4 Mapping Models to Formal Specifications

3.4.1 Mappings

Mathematically, there is no difference between models and formal specifications. Both are axiomatic descriptions of machines, and both may be either concrete or abstract in the sense of having transition rules or not. The formal top-level specifications used in the implementation paradigm discussed above

are almost always concrete, however, to facilitate the informal correspondence to the next stage, the source code.

Showing the consistency between a specification and a model means showing that any instance of the specification is also an instance of the model. This task is complicated by the fact that the specification and the model often have a different vocabulary. While the model talks about "subjects" and "objects," the specification may use terms such as "process," "buffer," "segment," etc.

It is necessary to map symbols representing sets and relations in the model to terms in the specification. Note that a single set in the model, like the set of objects, may correspond to the union of two or more sets in the specification, such as buffers and segments, or some other set definable in terms of the sets and relations in the specification. One then shows that the axioms of the model, when translated into the terms of the specification, are provable from the axioms of the specification. This is like constructing a valid interpretation of a model, except that the instance is another model instead of a particular relational structure (machine).

A mapping from model terms to specification terms is wrong if it fails to preserve the meaning behind the terms "subject," "object," "read access," "write access," "label," and others. Unfortunately, one cannot really tell from the specification itself whether the mapping preserves meaning. How do we know that "read" and "write" have not been interchanged? What would we do if the access modes in the specification were named "frob" and "grok"?

The fact that the axioms are preserved helps to some extent. For example, we must confirm that the labels in the specification form a partial ordering. However, if the partial ordering is a lattice, and its top and bottom are reversed (e.g., Unclassified switched with Top Secret) there would be no mathematical way to tell.

Ultimately, the only way to validate the mapping is to track it down to the hardware and machine language implementation. Even this does not really settle the question of what "read" and "write" were supposed to mean in the first place, since their properties were stated informally. Information-flow models such as the ones reviewed later address this question.

3.4.2 *Mapping Example*

As an example, we can show how to set up a mapping between the MAC system model, with the *-property as a state invariant, and the imaginary concrete model used above.

The first step is to set up a mapping between the sets and relations in the concrete model with those in the MAC system model. This requires us to identify the sets in the concrete model that serve as domains for the variables used above, and correspond to the elementary sets in the MAC system model.

Q is mapped to State, the set of states.

S is mapped to Process, the set of processes.

O is mapped to Segment, the set of segments.

M is mapped to {'read', 'write'}.

L is mapped to Class, the set of classification labels.

X is mapped to {get_read_access} × Process × Segment ∪ ....

This is not a complete mapping of sets, but it is enough so that we can go on to show how some functions are mapped. Consider the labelling function

$$\lambda : Q \to L^{S \cup O}.$$

It will have to be mapped to some concrete relation, which we shall also call λ, with the signature

$$\lambda : \text{State} \to \text{Class}^{\text{Process} \cup \text{Segment}}.$$

The closest we have in the concrete model is

$$\text{label} : \text{Process} \cup \text{Segment} \to \text{Class}.$$

We can define the concrete version of λ so that the label assignment is the same in every state; i.e., for each z of type Process or Segment,

$$(\lambda(q))(z) = \text{label}(z).$$

This equation defines the concrete version of λ. Similarly, we can map α with the equation

$$(\alpha(q))(p, i) = \text{access\_matrix}(q, p, i).$$

In effect, the abstract model functions have been added as an extension to the concrete model, by defining them in terms of the available functions there. We can also identify the sets in the concrete model with their abstract names; thus, Q = States, etc.

Once the mapping is complete, one shows consistency by proving that the axioms of the abstract model are satisfied. For example, we must prove the *-property. In formal notation, the *-property is

$$(\text{read} \in (\alpha(q))(s, o) \to \lambda(q)(o) \leq \lambda(q)(s))$$

$$\wedge\ (\text{write} \in (\alpha(q))(s, o) \to \lambda(q)(s) \leq \lambda(q)(o)).$$

When this property is expanded into concrete model terms using the equations above, it becomes

$$(\text{'read'} \in \text{access\_matrix}(q, p, i) \to \text{label}(i) \leq \text{label}(p))$$

$$\wedge\ (\text{'write'} \in \text{access\_matrix}(q, p, i) \to \text{label}(p) \leq \text{label}(i)).$$

But this is exactly the state invariant included as an axiom in the concrete model.

4. The Bell-LaPadula Model

4.1 Introduction

The Bell-LaPadula model has been influential in the development of secure systems with multilevel access-control policies. Perhaps this was because it was the first concrete MAC system model, and thus the first one suitable for the first stage of the implementation paradigm discussed earlier. It is also the reference of choice as the source of the *-property. The story behind the name "*-property" is that the authors couldn't think of a satisfactory name for what they recognized as an important axiom, so a "*" was left in place of a name to be supplied later.

The model evolved and expanded over several versions. We shall highlight the features of the first three briefly, then describe the Multics Interpretation in detail. There has been some recent debate surrounding the definition, purpose, and adequacy of the Bell-LaPadula model, stimulated by McLean's "System Z;" those issues will be touched upon.

In Volume I (Bell and LaPadula, 1973), the *-property and rules had not yet appeared. Any kind of access required that the subject dominate the object in security level. Security levels had classification and need-to-know components, but specific classifications were not named. The state had both a current access allocation (what we have called an access matrix) and an "access matrix" representing discretionary permissions; no axiom relating to the latter was specified.

Volume I (LaPadula and Bell, 1973) limited the access modes to read, write, append, execute, and control. It introduced a form of the *-property, with execute access viewed as a kind of read, and "write" access actually implying both read and write; append access was write-only. It had 10 transition rules. Rules for giving and rescinding discretionary access permissions tested control access.

Volume III (Bell, 1973) introduced an object hierarchy as a new component of the state. The hierarchy was used to do away with control access; a subject implicitly had control access to an object if it had write access to the parent object in the hierarchy. The hierarchy had to satisfy a compatibility property discussed below; this necessitated some changes in the rules. Subjects were given a current security level, distinct from, but dominated by, their maximum security level, leading to a change in the way the *-property was stated.

The "Unified Exposition and Multics Interpretation" (Bell and LaPadula,

1975) had a set of rules intended to be suitable as kernel primitives for a secure version of the MULTICS operating system, and it added the discretionary security property, which forced current accesses to be consistent with the permission matrix. The elements, relations, and axioms of this version will be given in a somewhat abbreviated form, and the rules summarized.

4.2 The Abstract Model

The abstract part of the model defines a kind of machine that we shall call a "BLP machine." A BLP machine has state set V, inputs R called *requests*, and outputs $D = \{\text{yes, no, ?}\}$ called *decisions*. Decision outputs are associated with transitions rather than states. A state has four components, (b, M, f, H), which will be described below along with other elements of the model.

As a reference monitor, a BLP machine has a set of subjects S, which is a subset of a set of objects O, and it has a set of access attributes $A = \{\text{r, e, w, a}\}$. Each state has an access set component, denoted with the symbol b, and representing current accesses as a set of triples (s, o, x) included in $S \times O \times A$.

As a MAC system, a BLP machine has a lattice L of security levels. Each level has two components: a classification from a totally ordered set C, and a subset of the set K of categories. Subsets of K are partially ordered by set inclusion, and the lattice ordering $\propto$ on L is induced as the direct product $C \times \mathscr{P}(K)$. That is, $(c, x) \propto (c', x')$ if $c \leq c'$ and $x \subset x'$ For example, (Confidential, {NATO}) $\propto$ (Secret, {NATO, NUCLEAR}).

Security levels are assigned to subjects and objects by another component of the state, symbolized f. An f-component is actually a triple (f_S, f_O, f_C), where

$f_S : S \to L$ is the subject (maximum) security level function,
$f_O : O \to L$ is the object security level function, and
$f_C : \mathrm{S} \to L$ is the subject current security level function.

The current security level is the one that plays a part in the *-property. The two levels are motivated by the idea that when a user logs in to a computer system, a process is created to communicate with the user's terminal and issue system commands. The process operates at a current security level requested by the user, and that level may be at or below the clearance of the user, which is recorded as the maximum level of the process. It is required that $f_C(s) \propto f_S(s)$.

There are two axioms relating current access to level assignments: the simple security property and the *-property. The simple security property appeared first in Volume I, and states that a subject can have read access only to objects at or below its maximum level.

Simple Security Property: For each state $v = (b, M, f, H)$,
if $(s, o, \text{r}) \in b$ or $(s, o, \text{w}) \in b$, then $f_O(o) \propto f_S(s)$.

The *-property has an exception built into it for subjects in a distinguished set S_T of "trusted" subjects.

*-*Property:* For each state $v = (b, M, f, H)$.
if $(s, o, \text{r}) \in b$ and $s \notin S_T$, then $f_O(o) \propto f_C(s)$;
if $(s, o, \text{w}) \in b$ and $s \notin S_T$, then $f_O(o) = f_C(s)$; and
if $(s, o, \text{a}) \in b$ and $s \notin S_T$, then $f_C(s) \propto f_O(o)$.

Two other components were added to the state to support discretionary access control. There is an access matrix $M : S \times O \to \mathcal{P}(A)$ whose elements represent access permissions rather than current access. (Actually, in the report (Bell and LaPadula, 1975), the subjects and objects were viewed as indexed by the positive integers, and M was a matrix with elements M_{ij}.)

An object hierarchy was introduced as a way of controlling the assignment and propagation of access permissions. It was motivated by the directory structure of Multics, though the model does not distinguish between non-directory objects and directories without subordinate objects. Formally, the hierarchy component H is a function on O into $\mathcal{P}(O)$, giving the set of subordinates of each object. H is a hierarchy in the sense that the directed graph induced on O, with edges from an object to each subordinate, is a forest, i.e., a set of rooted trees. A hierarchy is illustrated in Fig. 2.

The discretionary security property states that current accesses are restricted to accesses permitted in M.

Discretionary Security Property: For each state $v = (b, M, f, H)$,
if $(s, o, x) \in b$, then $x \in M(s, o)$.

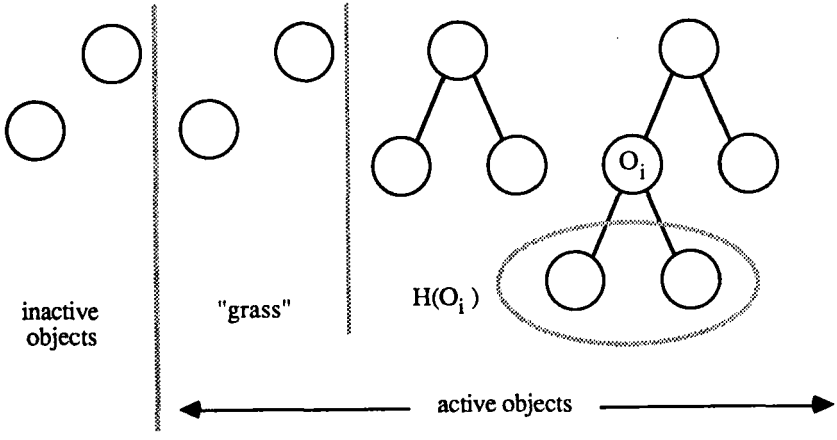


FIG. 2. A hierarchy.

The three state axioms given above, the simple security property, *-property, and discretionary security property, are considered the security policy for the abstract model.

4.3 Transition Rules

There are eleven rules, R1–R11. Each rule is a function on $R \times V$ into $D \times V$, giving the decision output and next state for each possible request and current state. Each rule is intended to handle a particular kernel request. If a rule Ri is not applicable to an input x, $Ri(x, v) = (?, v)$. Each rule refuses any request that would leave the system in a state violating the security policy. For such requests, the rule yields a value of (no, v). Acceptable requests yield a value of (yes, v') for some next state v'.

The 11 rules handle the following types of requests:

get-read, get-append, get-execute, get-write (four rules);
release-read/execute/write/append;
give-read/execute/write/append;
rescind-read/execute/write/append;
create-object;
delete-object-group;
change-subject-current-security-level;
change-object-security-level.

The get requests add an element to b, consistent with the three security policy axioms. The release request deletes an element from b.

The give request adds an access permission to M, and a rescind request takes it away. Inputs for these requests have two subject parameters—one representing the requestor, and one who will get or lose the access permission. These rules check that the requestor has write access to the parent object of the object involved in the affected access permission.

The create and delete requests cause objects to become attached to, or detached from, the active part of the hierarchy. A create request selects a (presumably inactive) object and augments H by adding it as a new child of a specified object (to which the requestor has write access). The active objects are those that are parents or have children, plus a few special isolated objects (called "grass"). The create rule does not actually check that the newly added object is inactive, though it should.

The newly activated object also receives a new security level. This suggests that there is more going on here, from a security point of view, than can be represented in an access-control model. An inactive object is supposed to be

erased, i.e., cleared of information. This consideration affects the appropriateness of implementations of the model, such as the need to preserve the meaning of "read" access from an information-flow point of view.

When an object is deleted, it is removed from the hierarchy, and so are all objects below it in the hierarchy; this makes all those objects inactive. At the same time, all subjects who have access to these objects lose it.

The rules preserve a property of the hierarchy called *compatibility*, credited to Walter *et al.* (1974a). A hierarchy is compatible if every subordinate object dominates its parent in security level. It is needed to prevent a covert channel for compromising information. If an object were below its parent in security level, a subject at the level of the parent could delete the object, and that action would be detectable by a lower-level subject who had access to the deleted object.

The rules for changing security levels do not affect the current access set *b*, but they require that the resulting state satisfy the policy axioms. Changing security levels is obviously the kind of activity that should be undertaken only with care. Volume II (LaPadula and Bell, 1973) states the *tranquility principle* on page 19: "the classification of active objects will not be changed during normal operation." This was stated as a consideration used in designing rules, but one that could be rejected as a matter of policy. The rule for changing object levels actually includes a special undefined test for "additional policy enforcement," which could decide upon "abnormal" operation.

When changing a subject level, if the change is a downgrade, one must assume either that subjects have no memory, or that any local memory of a downgraded subject is erased, in order to avoid a possible compromise. Even under these assumptions, there is still a covert channel, since a subject carries with it, in the current access set *b*, the record of which objects it has access to (Millen, 1984).

4.4 System Z and Tranquility

The security policy was expressed in the Bell-LaPadula model by three state axioms: the simple security property, the *-property, and the discretionary security property. There are a number of other axioms that are part of the context in which the security policy is stated, and which are equally part of the model: those expressing the lattice ordering on security levels, the structure of the hierarchy as a forest (with a proper definition of active objects), and the subject maximum level as the upper limit of its current level. But the rules, and certain properties that they satisfy, may be modified or replaced to suit the needs of various application systems. In particular, the tranquility principle and the compatibility property were not formally part of the abstract model.

Because the Bell-LaPadula model has played such an important role in the development of secure systems, especially those acquired by the U.S. Department of Defense, it is worthwhile to examine how well the model serves as a statement of requirements for security. If a system obeys the Bell-LaPadula security policy, is it really secure? This is part of the more general question of how one evaluates models; under what circumstances is a model satisfactory?

It is, of course, unreasonable to expect security to follow from a correspondence with an access-control model. The model only works within its level of abstraction; it is up to the implementor to make sure that the concepts such as "read" access are implemented as intended. Looking at the model as an abstraction of the implemented system, it should be a faithful representation. Still, one wonders whether the model has said as much as it could.

The lack of some suitably general and formal statement of the tranquility property is particularly disturbing, since downgrading an object is a quick and easy way to compromise information. As a graphic example of a system that is intuitively insecure and yet satisfies the Bell-LaPadula security policy axioms, McLean (1987) proposed "System Z." Based on the Bell-LaPadula abstract model, it has exactly one transition rule:

> When a subject s requests any type of access to an object o, every subject and object in the system is downgraded to the lowest possible level, permission is entered into the access matrix M, and the access is recorded in the current access set b.

A response by Bell (1988) argued that

> A model such as the Bell-LaPadula model that was constructed as an abstraction to allow analysis free of irrelevant detail never claimed to be a justification of "axioms" in a foundational sense, nor did it claim to capture all the facets of intuitive-security.

He goes on to point out that a universal downgrading rule as in System Z is not necessarily insecure from an intuitive point of view. It may be invoked in a situation where the computer system has been captured by an enemy and all objects are erased. The erasure is not expressible in an access-control model, but it is a requirement for the implementation, just as individual objects must be erased before they are activated by the create-object rule.

Erasure is evidently an awkward subject for access-control models. There is

a way of handling it that is better suited to the level of abstraction of such models, though one still needs to think about how to implement it. Instead of permitting objects to alternate between active and inactive (erased) states,

$$\text{active} \leftrightarrow \text{inactive},$$

let us assume that there is an infinite pool of objects, so that each object need only be active once. An object just goes through three states:

$$\text{never used} \rightarrow \text{active} \rightarrow \text{dead}.$$

The fact that a newly active object contains no information derived (via accesses) from any other object is then obvious from the model, and no special instructions about erasure are needed. An implementation that carries forward the spirit of this model will still reclaim the space allocated to dead objects, but it will treat each newly activated object as conceptually new. The uniqueness of each new object is reflected by assigning it a previously unused "unique identifier," as is done in SCOMP (Fraim, 1983) and PSOS (Neumann *et al.*, 1977).

In some applications, there are reasons for downgrading or otherwise changing the level of objects without erasing them. This should only be done on the request of a privileged subject. McLean (1988) has suggested a model in which the level of each object o can be changed only at the request of a defined set of subjects $c_O(o)$. A similar function can be defined for subjects. Inputs are of the form (s, r) where s is the requesting subject and r is a request. Limited tranquility is then expressed as an axiom, saying that a transition, due to input (s, r), that changes the level of an object o, is possible only when $s \in c_O(o)$. If every subject is associated with a set of users (people), and there is some way (in the implementation) of ensuring that inputs from a subject are actually authorized by its users, then one can choose c_O in such a way that it represents more complex policies such as n-person control.

It is obvious that downgrading objects is a questionable operation that should be performed only under special conditions, but it may be less obvious that upgrading objects can also cause problems. Of course, it is undesirable and usually against policy to overclassify information by marking it at a higher sensitivity level than it deserves, but upgrading can also compromise information through a covert channel. When an object is upgraded, lower-level subjects that had read access to it in the past will lose that access. If the upgrade was performed at the request of a higher-level subject, this a a way for higher-level subjects to affect lower-level subjects. To avoid any possibility that a high-level subject might covertly signal information to a lower-level subject, upgrades are either not permitted, or permitted only at the request of subjects at the original object level.

4.5 Trust and Integrity

4.5.1 *Trusted Subjects*

How is it that we can trust certain subjects with risky privileges, such as downgrading objects or having write access to a lower-level object? Is "trust" meaningful as a modelling concept, in an environment with faulty software and Trojan horses? Wasn't the *-property invented precisely because user programs could not be trusted?

The answer is that "user" programs are ordinarily not trusted. Processes (subjects) are trusted only when they execute trusted software that has been examined as carefully as the operating system kernel software. The kernel protects this software in the same way that it protects itself, by refusing any attempt by any unauthorized process to gain write access to the memory containing the trusted software. Processes become trusted only by the action of the kernel, which initiates their execution at an entry point of a trusted program.

4.5.2 *Biba's Integrity Model*

Kernel protection of "trusted" software applies only to software that the kernel knows about as part of the design of the system. There is also a need, in many applications, to protect some programs or data that may be entered into the system by ordinary users at any time. This general concern is referred to as protecting the *integrity* of objects, and it is addressed through methods for preventing unauthorized write access to the protected objects.

Discretionary access controls can be used to limit write access, but they work only if all subjects who have write access are trusted, and all subjects who can give away write access will give it only to trusted subjects. This means that if a Trojan horse can get either write access to a protected object, or the ability to give it away, then the protection is a failure. In practice, this often means that all the programs available to a user must be trusted.

Biba (1977) realized that nondiscretionary access controls could also be used for integrity, even though they were originally intended merely to prevent compromise of information. He also discussed discretionary integrity controls, but we shall focus on the label-based controls here.

Subjects and objects are labelled with integrity levels. Biba suggested "Crucial," "Very Important," and "Important" as integrity classes, but any partially ordered set can be used. If we think of a high-integrity level, e.g., Crucial, as dominating a low-integrity level, e.g., Important, the information flow policy for these levels is the opposite of that for sensitivity levels. Information flow from one entity to another should be allowed only when the

destination carries an integrity level *dominated by* that of the source. Information can lose its integrity; it can never gain in integrity.

In Biba's model, subject can *observe* or *modify* objects, and *invoke* other subjects. Invocation is meant to be interpreted as interprocess communication or procedure calls (into a different protection domain). Invocation causes information, in the form of a message or parameter values, to flow from the invoking subject to the invoked one.

Four different access control policies were proposed by Biba. The simplest and best remembered is the *strict integrity* policy, which permits a subject

- Observe access only to objects of a higher or equal integrity level.
- Modify access only to objects of a lower or equal integrity level.
- Invoke access only to subjects of a lower or equal integrity level.

In the strict integrity policy, integrity levels do not change.

The other three policies allow various relaxations of the axioms of the strict integrity policy. They are: a *low-water mark* policy, in which a subject can observe objects of lower integrity level, but its own integrity level is reduced accordingly; a *low-water mark for objects* policy, a low-water mark policy in which a subject can also modify objects of a higher integrity level, but the integrity level of those objects is immediately reduced; and a *ring* policy, in which observation is unconstrained.

The two low-water mark policies still enforce the strict-integrity state axioms, but only at the cost of changes in the integrity level assignment. The ring policy works only when it can be assumed that a subject of high integrity is executing a program of high integrity, which is not misled by observing objects of lower integrity. The problem here is that executing a program is a form of observe access; so a high-integrity subject might be a process executing a low-integrity program, which is inconsistent with the required assumption. The ring policy would be more effective if execute access could be distinguished from observe access (and that distinction could be enforced in the implementation).

4.5.3 Strict Integrity is Free

If one leaves out invoke access, the remaining access restrictions for strict integrity an observe and modify access are the dual of the *-property. Interpreting "observe" as "read" and "modify" as "write," the only difference is that the directions of the partial ordering are reversed. This suggests that a mechanism for enforcing the *-property can be extended to enforce strict integrity without much difficulty. In fact, in many cases the *same* mechanism will work, and it can enforce both the *-property for compromise protection and strict integrity simultaneously.

The idea is to redefine the label set. If one has a partially ordered set of sensitivity levels, say C, and a partially ordered set of integrity levels, say I, one can define a new set of labels $L = C \times I$, with a partial ordering defined as follows:

$$(c, i) \leq (c', i') \text{ if } c \leq c' \text{ and } i \geq i'.$$

Thus, if the *-property is enforced with these labels, a subject can have write access to an object only if the label of the subject is dominated by that of the object, and this means that the sensitivity level of the subject is dominated by that of the object, while the integrity level of the subject *dominates* that of the object. This is just what we wanted for strict integrity, and it works similarly for read access. From an abstract model point of view, nothing new has been added. From an implementation point of view, the only concern is having enough bits in a label to represent both levels. Note that if label comparison is implemented by arithmetic comparison, there is no need to change the comparison test. Simply use zero to represent the highest integrity level and use the highest number to represent the bottom integrity level.

In practice, the main problem has been figuring out what integrity levels to use, and what they mean. An arbitrary list, like the Critical to Important range in Biba's report, is not likely to correspond to any useful or mandated policy. Using the classification range Top Secret to Unclassified is a real mistake, since it is confusing if the integrity class does not match the sensitivity class. On the other hand, the system is unusable if the same class is used for both, since the reinterpreted *-property will constrain a subject to access only objects of exactly the same class.

One easy and constructive way of using an integrity level is simply to distinguish between "trusted" and "untrusted," with the "trusted" label applied only to objects containing software believed to be trustworthy. Or, in an environment with mutually suspicious users, have a label per user, like "trusted-Smith," "trusted-Jones," etc., which are mutually incomparable but all dominating "untrusted." Strict integrity will then provide protection against such threats as Trojan horses and viruses.

4.5.4 *Type Enforcement*

Strict integrity may be easy to implement, but it does not address all integrity needs. According to Clark and Wilson (1987), in a data processing environment there is often a need to ensure that certain "constrained data items," or CDI's, are manipulated only by specified "transformation procedures," or TPs. A TP is entrusted to read a CDI of one type and create an output CDI of a different type. This sort of processing is essentially the same as an "assured pipeline" as described by Boebert and Kain (1985). Pipelining is a

special case of a type enforcement scheme in which each program is restricted to have read objects only of specified data types, and write access only to objects of specified data types.

Boebert and Kain make the point that type enforcement cannot be implemented with a nondiscretionary policy using partially ordered labels. For, suppose that the pipeline is two steps long, e.g.,

$$A \rightarrow \text{TP1} \rightarrow B \rightarrow \text{TP2} \rightarrow C,$$

where A, B, and C are CDI types. Suppose that integrity labels are assigned to A, B, and C, and also to the objects containing the programs TP1 and TP2, in such a way that the reads and writes in the pipeline are permitted by strict integrity. This would imply that A and TP1 had a greater or equal integrity level than B, and that B and TP2 had a greater or equal integrity level than C. What, then, is to prevent TP1 from writing into C as well? Not the *-property.

It is possible to turn the partial ordering around so that all of the reads and writes in the pipeline tend to increase, rather than decrease, the integrity level. In that case, the *-property would refuse the needed accesses. But, now we can say that subjects executing TP1 and TP2 are *partially trusted* (this term comes from Lee (1988)), and will be given a special dispensation sufficient to accomplish their necessary accesses. As a policy, this is certainly another way of accomplishing the effect of a pipeline, but there are as many flavors of partial trust as there are TPs; this is a complex policy.

5. Database and Network Models

5.1 Database Management System Models

Most of the work in secure database management systems (DBMS) has been done in the context of *relational* systems, in the sense of Codd (1970). A relational database is a set of relations. In a mathematical context, each relation is a subset of a cartesian product of domains; its elements are tuples. In a DBMS context, domains are often called *fields*, and the tuples are referred to as *records*. The components of an individual record are called *data elements*. See Fig. 3 for an illustration.

Relations in a DBMS must have a *key* field or fields. By definition, the data elements in a key field identify records, in the sense that there is at most one record with a particular data element in the key component. Sometimes two or more key fields taken together are needed to constitute a key. There may be more than one set of fields that satisfy the properties of a key; one of these is selected as the *primary* key.

DBMS models differ from one another primarily in the way they slice up relations into objects, which has implications for the way labels are assigned.

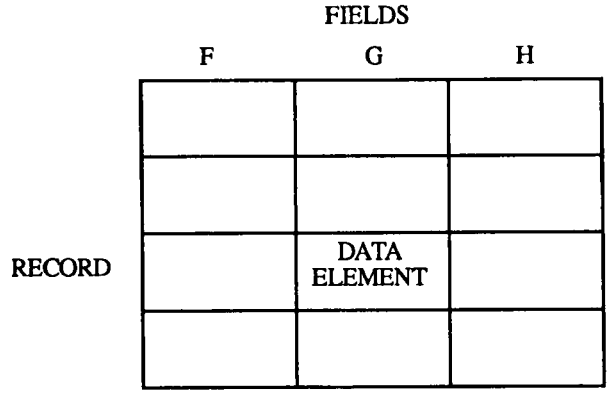


FIG. 3. A DBMS relation.

Labels may be assigned by field, by record, by data element, or by relation. These choices have consequences for the way the DBMS is implemented. The finer the granularity of objects, the less likely it is that a general-purpose secure operating system kernel will provide both full data protection and efficient service, and the more special-purpose trusted code is likely to be added.

One of the earliest approaches, by Hinke and Schaefer (1975), assigned classifications by field. They found that they needed axioms saying that the (primary) key fields of a relation all had the same classification, and that all other fields had a classification dominating that of the key. The reason for this is that, in order to read a data element of a record, the DBMS implementation must find the right record first, using a search procedure that reads the key field. Entering and updating records in this system is complex for a relation having fields at different levels, since the subject that enters a high-classification data element in its proper field cannot also enter a data element into a low-classification field, and *vice versa*.

The I. P. Sharp model (Grohn, 1976) assigned protection levels by relation. Their protection levels, incidentally, included an integrity level, in accordance with the strict-integrity model mentioned in the previous section. With a protection level on an entire relation, it is still possible to simulate the assignment of levels by field. The trick is to create a separate relation for each non-key field, each one having its own copy of the key, and classified at the level desired for the non-key field. A subject at the level of one of the higher-level fields could then use relational DBMS operations to assemble the information from all fields (relations) at its level and below.

A Naval DBMS model (Graubart and Woodward, 1982) assigns protection levels at a data element granularity. However, records, fields, and relations and the entire database are viewed as containers and may each have their own

default security level (DSL). The actual level of a data element is chosen from several levels: the DSL associated with its unique location as a data element, and the DSLs of all containers it belongs to. Some of these DSLs may be unspecified (but the database always has one). When there is a conflict, because there are two or more applicable DSLs that are specified and different, a priority scheme is used to determine the final level.

Another approach to the assignment of labels is to attach them to views, as suggested by Denning *et al.* (1987a). A *view*, in a relational system, is a formula for constructing a new relation from one or more base relations. The base relations are the ones in which data is actually stored; views are stored only as formulas. Users cannot access base relations directly; they may only see and operate on views.

Classifying views permits a more flexible approach to some policy issues regarding the classification of data in a database. For example, data might sometimes be classified by value. Suppose a database on private airline flights has a relation showing the principal passenger of each flight. Records showing passengers from a specific list, e.g., the President or foreign dignitaries, might be assigned a higher sensitivity level than others. One can also address the *aggregation* problem, referring to the fact that a large enough accumulation of records can become more sensitive than any of the individual records.

Data entered into the system is assigned a level on a data-element basis, using rules called *classification constraints*; these can be expressed as views also. Access views—the ones users see—normally receive a level just sufficient to cover the levels of the data elements that must be assembled to construct a view instance. Special policy considerations may result in a different level, however. With this sort of policy, the operations for constructing views (using relational operators) and assigning them levels must be trusted.

In most of the policies discussed above, objects were assigned labels on the basis of their location attributes. The location of a data element is determined by identifying the relation it belongs to, the key of the record it is in, and which field it fills. But, in some systems, this location information does not uniquely determine the label of the data. For example, the label might be affected by the value of the data element, or the label might have been set to different levels depending on the level of the subject that updated the data element. In such systems a problem arises.

If the location information does not determine the label of the data, then certain data might exist for a location without being visible to lower-level subjects. This means that lower-level subjects might update the location without being aware of a conflict. It also means that higher-level subjects might signal information to lower-level subjects with an update that makes the contents of the location invisible.

The solution chosen for the SeaView model (Denning *et al.*, 1987b) is

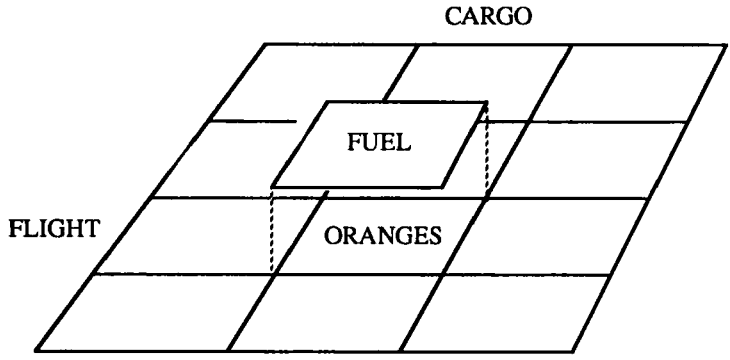


FIG. 4. Polyinstantiation.

polyinstantiation. Multiple versions of a data element, record, or relation are created as necessary to reflect updates at all sensitivity levels. When a flight record originally shows oranges as a cargo, and someone updates it to show nuclear fuel, an uncleared subject will see only the version of the cargo that says oranges, while a higher-level subject will see the nuclear-fuel entry (see Fig. 4.) This prevents the two problems with updates, since an uncleared subject cannot cause the nuclear fuel to be left behind by changing oranges to apples, nor can a higher-level subject covertly signal information by affecting the lower-level view of the relation.

5.2 Network Models

Multilevel security in networks is a recent phenomenon. Very few examples exist, and it is difficult to say whether formal models had an important role in their development. Current approaches to multilevel modelling of networks are in flux, and it seems too early to draw conclusions on how best to do network modelling. Two published examples will be mentioned to give a flavor of what happens when multilevel access-control considerations are applied to networks.

The abstract model for SNet (Glasgow and MacEwen, 1987) sees a network as a medium for transmitting labelled messages between subjects. The security properties of the network ensure both that sent messages are labelled with the label of the sending subject, and also that received messages are delivered only to subjects whose label dominates that of the message. SNet subjects are intended to represent hosts or terminal concentrators. Some are trusted; a trusted subject is permitted to change its current level to any level below a

specified maximum. The network has a global state, consisting of two histories for each subject: a transmit history and a receive history, each of which is a sequence of messages. The network state changes as the result of a send or receive event by some subject, which extends that subject's history.

Besides the axioms relating to labels, the SNet model has other axioms stating that messages are not misdelievered, and that every received message was sent. Messages include sender, receiver, and data components, so these axioms ensure that message data has not been relabelled or substituted from another message while in transit. The SNet work also includes a formal specification showing more of the network structure, and a proof that it satisfies the model.

McHugh and Moore (1986) have a model they describe as a simplified version of the Bell-LaPadula model. The subjects are network hosts, and the objects are datagrams. Instead of separate send and receive events, their system has *communication* events. A communication event is a triple (s, o, s') where s is the sending subject, s' the receiving subject, and o is a datagram. It is secure if the classification of o dominates the clearance of s, and the clearance of s' dominates the classification of o. The network state is a set of communication events—the ones that have taken place so far—and it is secure if its elements are all secure. There is also a discretionary aspect to the policy, in that only certain pairs of subjects are authorized to share communication events. This policy has been shown to hold for a formal specification written in Gypsy.

These models both interpret subjects as hosts and treat the network as a single large machine. Taking a host as a subject is not unreasonable for nondiscretionary access control purposes, as long as trusted, multilevel hosts have been shown to deserve their privileges. Taking a network as a single large machine, however, is only the first step in a process that decomposes the network into its components and examines the role of each component. The most productive way of doing so, from a formal modelling point of view, has yet to be seen.

6. Information Flow Models

6.1 Introduction

There are ways to compromise information in a computer system that cannot be understood solely from access control considerations. If examination of access control mechanisms in a computer system design is like using

a magnifying glass, current research in computer security modelling has the objective of constructing an electron microscope. This survey of multilevel modelling will conclude with a summary of these new directions in research.

A mechanism by which a process operating at a high sensitivity level can send information to a lower-level process, in spite of an access control policy, is termed a *covert channel.* Some covert channels arise from the way the system is implemented: Lampson (1973) provides an example of a timing channel, in which a process communicates to others by varying the time it requests for computation. Other channels can be recognized in an abstract design specification of a system, even in a concrete model. We noted a channel inherent in one of the Bell-LaPadula transition rules for Multics, and we saw that polyinstantiation in a secure relational database was motivated partly by covert channel concerns.

There have already been a number of models aimed at defining information flow in abstract machines, with sufficient precision so that covert channels can be explained and detected. In these models, we can state axioms to the effect that no information flow occurs from a subject to another, except when the security labelling would permit. There has been some effort to develop tools and techniques based on these models, for detecting covert channels in system specifications. Some fairly recent applications of these methods are discussed by Haigh *et al.* (1986) and Benzel (1984). Covert-channel analysis is presently difficult, but the models and tools are still being developed.

Information-flow models share the philosophy that information flow is related to inference: if one subject can, by observing outputs available to it, deduce something about inputs from another subject, there has been some information flow. Conversely, if there is no information flow, the first subject's outputs would be independent of the input from the other subject. This idea was originally suggested by Jones and Lipton (1975), for computations rather than machines. One direction of development from the computation idea was to look at the computations occurring in high-level-language programs, due to individual statements, subroutines, or the entire program. This led to the definition by Cohen (1978) of strong dependency between variables in a program, and to syntactically-based analysis techniques as given by Denning and Denning (1977). Millen (1978) expressed information compromise from one state variable to another due to inference in a nondeterministic machine, and there was a model due to Feiertag, *et al.* (1977) that formulated a policy for deterministic machines that prevented information flow from inputs at a high level to outputs at a lower level. These early approaches were surveyed by Landwehr (1981).

There have been some significant advances since then. The next step was a paper by Goguen and Meseguer (1982), defining a notion called noninterference, which was a generalization of the Feiertag model.

6.2 Non-interference

6.2.1 *Definitions*

Non-interference was defined in the context of a machine composed of

- A set S of states, with an initial state $s_0 \in S$.
- A set U of users (or subjects).
- A set C of commands (or operations).
- A set O of outputs.

together with functions

- do: $S \times U \times C \to S$.
- out: $S \times U \to O$.

We may think of $U \times C$ as the set of inputs for this machine. Inputs are thought of as coming from particular users, and in each state there is an output available to each user.

Terminology. Let $(U \times C)^*$ be the set of sequences of inputs in $U \times C$. If $w \in (U \times C)^*$, we can start the machine in its initial state and apply the inputs in w successively, leaving the machine in some state which we shall denote by $[w]$. Let $[w]_u = \text{out}([w], u)$.

Given an input sequence w and a user u, define w/u as the subsequence of w obtained by deleting all inputs of the form (u, c) for some c. (This notation comes from Rushby (1985).)

A user u is *non-interfering* with user v if, for all $w \in (U \times C)^*$,

$$[w]_u = [w/v]_u.$$

We write $u \not\rightarrow v$ as an abbreviation for the statement that u is non-interfering with v.

This says that the final output to u would be unaffected if all inputs from v were deleted. Previous outputs to u would also be unaffected, since they are the final outputs of shorter input sequences. It is claimed that non-interference precisely captures the notion of information flow, in the sense that there is no information flow from u to v if and only if u is non-interfering with v.

Goguen and Meseguer also define non-interference between groups of users. First, if w is an input sequence and A is a set of users, define w/A as the subsequence of w with inputs from all users in A deleted. Then, for $A \subset U$ and

$B \subset U$, A is non-interfering with B (written $A \not\rightarrow B$) if, for all $v \in B$,

$$[w]_v = [w/A]_v.$$

A multilevel security (MLS) policy can be stated as soon as we add a labelling function,

$$\text{level}: U \rightarrow L,$$

where L is a partially ordered set of sensitivity levels. Goguen and Meseguer's policy states that the users at or above one level cannot interfere with users at or below a second level, if the second level does not dominate the first.

MLS1: Let $x \in L$ and $y \in L$ such that $x \not\leq y$.
Then $\{u \mid \text{level}(u) \geq x\} \not\rightarrow \{v \mid \text{level}(v) \leq y\}$.

Rushby (1985) states the multilevel security policy in a different form, for pairs of users:

MLS2: Let $u \in U$ and $v \in U$ such that $\text{level}(u) \not\leq \text{level}(v)$. Then $u \rightarrow v$.

It is not hard to show that MLS1 and MLS2 are equivalent. First, suppose that MLS2 is true, and let $x \in L$ and $y \in L$ such that $x \not\leq y$. Let $A = \{u \mid \text{level}(u) \geq x\}$. Choose v such that $\text{level}(v) \leq y$. Note that if $u \in A$, then $\text{level}(u) \not\leq \text{level}(v)$. By MLS2, if $u \in A$, then $u \not\rightarrow v$. Let $A = \{u_1, \ldots, u_n\}$. Then, for any input sequence w,

$$[w]_v = [w/u_1]_v = [(w/u_1)/u_2]_v = \cdots = [w/\{u_1, \ldots, u_n\}]_v = [w/A]_v.$$

Thus, MLS2 implies MLS1.

Now, suppose that MLS1 is true, and let $u \in U$ and $v \in U$ such that $\text{level}(u) \not\leq \text{level}(v)$. Again, let $A = \{u' \mid \text{level}(u') \geq x\}$. Let $x = \text{level}(u)$ and $y = \text{level}(v)$. By MLS1, $A \not\rightarrow \{v' \mid \text{level}(v') \leq y\}$. This gives us

$$\begin{aligned} [w]_v &= [w/A]_v && \text{by MLS1} \\ &= [(w/u)/A]_v && \text{since } u \in A \\ &= [w/u]_v && \text{by MLS1 again.} \end{aligned}$$

Thus, $u \not\rightarrow v$, showing that MLS1 implies MLS2.

6.2.2 *Unwinding*

If we agree that the non-interference MLS policy is a satisfactory definition of nondiscretionary security, there is still a practical problem: showing that a formal specification is consistent with it. The definition of non-interference in

terms of arbitrary input sequences is not easy to deal with. The unwinding theorem of Goguen and Meseguer (1984) expressed non-interference equivalently as a property that could be tested for each state transition. This brought it within reach of standard proof techniques for formal specifications.

The unwinding theorem will be presented in the somewhat simplified form given by Rushby (1985). The key to unwinding is to notice that each user has a limited view of the machine, determined by the outputs available to that user. Two states are equivalent for a user if they cannot ever be distinguished by that user, on the basis of subsequent outputs. One user is non-interfering with a second user if state transitions caused by the first user go to another state that is equivalent for the second user.

An equivalence relation $\equiv$ on the set of states S is a *congruence* with respect to a user $v \in U$ if

- $s \equiv t$ implies $\text{out}(s, v) = \text{out}(t, v)$, and
- $s \equiv t$ and $u \in U$ and $c \in C$ implies $\text{do}(s, u, c) \equiv \text{do}(t, u, c)$.

Unwinding Theorem: $u \not\rightarrow v$ if and only if there exists a congruence $\equiv$ with respect to v such that, for all $c \in C$ and $s \in S$ reachable from the initial state,

$$\text{do}(s, u, c) \equiv s.$$

The proof is given, in different forms, by Goguen and Meseguer (1984) and Rushby (1985). The proof that the existence of the congruence implies non-interference is accomplised by induction on the length of an input sequence. The proof that non-interference implies the existence of a suitable congruence is routine once the congruence is constructed. Since any reachable state can be expressed as $[w]$ for some w, define $[w] \equiv [w']$ if for all input sequences z, $[wz]_v = [w'z]_v$.

6.2.3 *Applying Unwinding to Multilevel Security*

Haigh *et al.* (1986) showed how the unwound formulation for non-interference could be used to check whether the SAT (Secure Ada Target) system, as specified by a concrete model in Gypsy, is free from covert channels. In attempting to prove that the MLS policy was satisfied, the proof failed, and by examining the reason for the failure they discovered a covert channel.

The proof method is to identify a good candidate equivalence relation on the states, for each subject, and try to show that it is a congruence, and also that it satisfies the unwinding condition for each pair of subjects u, v with $\text{level}(u) \not\leq \text{level}(v)$. If the proof succeeds, the MLS policy (MLS2) holds. If the proof fails, it does not necessarily mean that the MLS policy fails; it might only mean that they chose the wrong congruence relation. Nevertheless, if a failed

proof leads to the discovery of a covert channel, the effort has been worthwhile.

The candidate congruence relation was constructed by identifying the "subject view" of the state for each subject. The subject view consists of those state components that could eventually affect values returned to the subject. Two states with the same values in the components belonging to a certain subject view are equivalent for that subject. One way of assigning subject views is to try to associate a sensitivity level with each component or sub-component of the state. Those components at or below the sensitivity level of a subject are in its view.

As a practical method for covert channel analysis, there are two drawbacks to this approach: one is the skill required to find a good congruence relation, and the other is the skill required to trace the cause of a failure to a covert channel. It might be argued that something like clairvoyance is required, rather than skill; but an understanding of the system architecture is probably sufficient to do the job. In this respect, there are no clearly superior methods for performing covert channel analysis.

Since the non-interference approach is so general, why limit it to convert-channel analysis? Are access-control models still needed? Access-control models are useful because they provide understandable design guidance, the system is expected to enforce its access-control policy, and the mechanisms for access control are clearly visible in the machine architecture. The non-interference MLS policy, on the other hand, gives no design guidance, and proofs of it generally fail because real systems have covert channels. Some may be eliminated when they are found, but others are not serious enough to remove.

6.3 Restrictiveness

6.3.1 Nondeterministic Systems

Non-interference has one significant limitation: it applies only to deterministic machines. Consequently, it is not applicable to many multiprocessor systems and networks, since they are often nondeterministic. Nondeterminacy arises from the unpredictability of delays that occur in distributed systems. There are two sources of delay: propagation of signals within a component, and propagation of messages between components. Because of these delays, networks are subject to race conditions; a component may behave differently, depending on which of two messages reaches it first, and either way is possible.

Some work has been done by McCullough (1987, 1988a, 1988b) on generalizing interference in the context of an event-system model of com-

putation. Consider a set E of events, which correspond to the primitive actions done to or by a system. Of these, some are input events, others are output events, and the rest are internal. Let I be the set of input events and O the set of output events. A system will be characterized by the set of event sequences that are possible for it.

Formally, a system is a quadruple (E, I, O, T) where I and O are disjoint subsets of E, and T is a subset of E^*, the set of finite sequences of elements of E. T is called the set of traces, and it satisfies two axioms:

- *Event Separability*: If $t \in T$ and s is an initial subsequence of t, then $s \in T$.
- *Input Totality*: If $t \in T$ and $i \in I$, then $ti \in T$.

Event separability reflects the idea that a system might have been stopped at any time, so whatever events have happened up to any earlier moment constitute a possible trace. Input totality says that inputs cannot be prevented from coming at any time, and show up in the trace, though the machine may ignore them.

It should be clear that event systems can represent either deterministic or nondeterministic machines. One way of representing a deterministic machine, for example, is to record the entry into a state as an internal event. Ignoring outputs for the moment, the traces of a deterministic machine would then have the form

$$q_0 i_1 q_1 i_2 \ldots,$$

where each triple $q_n i_{n+1} q_{n+1}$ must be consistent with the transition function. Input totality must be recognized by adding other traces in which extra inputs have been added, as $q_n i_{n+1} i' i'' q_{n+1}$. The first or last input between states would be the one responsible for the transition, as a matter of convention.

It is not hard to come up with a version of non-interference that is plausible for event systems. First, introduce a set U of users, and associate inputs and outputs with users. Then we might say that a user u is non-interfering with another user v if the set of possible outputs to v is unaffected by deleting the prior inputs from u. If one alters a trace by deleting inputs from u, the resulting event sequence is not necessarily a trace, but one can find another trace with the same inputs, in which the final output to v is unchanged.

6.3.2 Composability

The problem with this, and other plausible generalizations of non-interference, is that it is not robust with respect to a very important construction: the act of connecting systems together into composite systems, or networks. The ability to compose systems and retain their security

properties is significant because such connections are used to

- Create nondeterministic systems from deterministic ones.
- Create networks.
- Create complex systems from simpler subsystems.
- Represent the interactions of trusted with untrusted processes.

McCullough (1988b) gives an example that illustrates the technical difficulty of generalizing non-interference in a composable way. Suppose we have two event systems A and B with the following behavior:

A has inputs and outputs associated with user u (which we think of as a high-level user), and two specific types of output associated with user v (the low-level user). One output to v is a "stop-count" signal, which occurs nondeterministically at any time. The next output to v is the parity (odd or even) of the total number of u's events, both inputs and outputs, that occurred prior to the stop-count output. Inputs and outputs belonging to u occur nondeterministically without restriction.

A typical trace for system A might be pictured on a vertical time line as in Fig. 5, where u's events are represented by dashed arrows and v's events with solid arrows. B is identical to A except that its stop-count signal is an input rather than an output. See Fig. 6.

It is plausible to say that u is non-interfering with v in both system A and system B. The reason is that v's parity output can be either odd or even, regardless of the number of inputs from u, since additional outputs to u might possibly be generated and change the count.

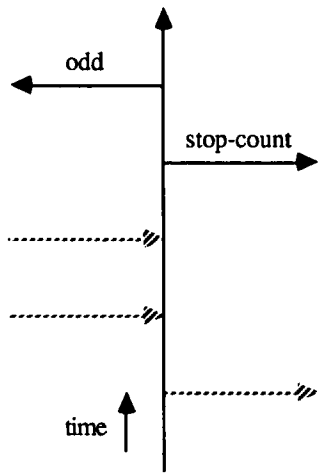


FIG. 5. A trace of System A.

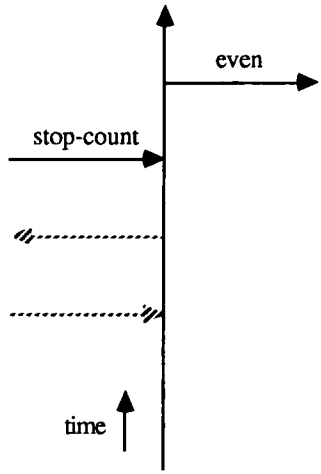


FIG. 6. A trace of System *B*.

We can connect *A* and *B* together into a composite system by taking the stop-count output from system *A* and feeding it into system *B* as a stop-count input. There should be nothing insecure about this connection, since both events belong to the same user, *v*. In the network, the stop-count signal has become a single event, which is neither an output nor an input, but rather an internal event. Also, *u*'s outputs from *A* are fed into *B* as inputs, and *u*'s outputs from *B* are fed into *A* as inputs. *A* still gets other external inputs, but *B* does not. We assume that no two events are simultaneous, so that events will appear in a discernible order in the traces of the composite system. A typical trace of the composite system is pictured in Fig. 7.

The reader is invited to check that the two parity outputs to *v* emitted from the two component systems permit *v* to determine whether *u* has entered an even or odd number of inputs. The number of inputs is odd if the parity outputs disagree, and even if they agree. Consequently, *u* is not non-interfering with *v* in the network. For, when *u* had an odd number of inputs, deleting them changes the overall parity, forcing a change in one of the two outputs to *v*.

6.3.3 *Restrictiveness and Multilevel Security*

McCullough then proceeded to define a new, stronger security property called *restrictiveness* that is much less obviously a generalization of non-interference, though it coincides with non-interference on deterministic systems. Restrictiveness was then shown to be preserved when systems were composed in such a way that labels on events are matched.

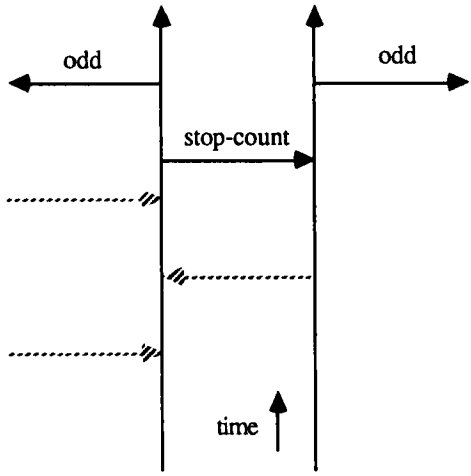


FIG. 7. A trace of the composition of System A and System B.

Terminology. If $s \in E^*$ is an event sequence, and $F \subset E$ is a set of events, $s \mid F$ is the subsequence of s consisting of just those events in F. Also, let ϵ represent the empty sequence. The letters a, b, c, etc., represent event sequences.

A set $F \subset E$ of events is said to be *restrictive* if the hypotheses

- $abc \in T$ (abc is a trace).
- $b, b' \in I^*$ (b and b' are input sequences).
- $b \mid F = b' \mid F$ (b and b' agree on F).
- $c \mid (I - F) = \epsilon$ (c has no non-F inputs).

imply the existence of $c' \in E^*$ such that

- $ab'c' \in T$ ($ab'c'$ is a trace).
- $c' \mid F = c \mid F$ (c and c' agree on F).
- $c' \mid (I - F) = \epsilon$ (c' has no non-F inputs).

Roughly speaking, any change in non-F elements of a trace segment b of inputs can be repaired by changing non-F elements of the following part of the trace. This conveys the idea that non-F inputs, and the users responsible for them, are non-interfering with F events, and the users who can observe them.

To get a definition of multilevel security, introduce a partially ordered set of levels L and a function level: $E \to L$. Note that events rather than users are

given levels, and that all events, including internal events, receive levels. An event system together with the level structure is called a *rated* event system. If $x \in L$, let $\text{view}(x) = \{e \in E \mid \text{level}(e) \leq x\}$, the events of level at or below x.

A rated event system is *multilevel secure* if, for all $x \in L$, $\text{view}(x)$ is restrictive.

It is shown in McCullough (1988a) that restrictiveness, and hence multilevel security, is *composable*, in the sense that if it holds for two systems A and B, it holds for a composite system in which outputs from either system have been merged with equal-level inputs of the other system. Other work comparable to the development of non-interference has also been done. In McCullough (1988b) there is a state-machine characterization of restrictiveness similar to the unwinding theorem for non-interference, and there has been some effort to apply it to a real system (Casey *et al.*, 1988). More work still needs to be done to check whether some simpler or weaker definition of multilevel security in nondeterministic systems is possible, and to find practical ways of applying it to detect covert channels or guide system design.

7. Conclusion

The common feature of the models we have been discussing is the use of sensitivity labels to restrict information flow. We have seen that, because of the nature of information flow, labels ought to be partially ordered, and it is often convenient to assume that they form a lattice.

When an information flow policy is implemented with an access-control mechanism, or reference monitor, the result is a mandatory access-control system that restricts access according to the *-property. A MAC system has the important advantage that it provides protection against Trojan horses, assuming that the privileged programs that set labels or perform other trusted functions are not themselves Trojan horses. Furthermore, there are computer architectures that support this kind of policy in a simple, understandable way.

MAC system models have been used with some success to help design secure computer systems. There is room for disappointment that a rigorous procedure cannot be followed, in practice, from a policy model all the way to verification of microprograms. Yet, there is evidence that taking the first steps rigorously, from a model to a formal specification, has resulted in better designs and has found bugs that might otherwise have taken longer to discover.

It is straightforward and practical to prove that a formal specification is consistent with a model, but the correspondence is relative to a particular mapping. Successful mappings are not unique; the right one must exhibit an appropriate interpretation of the model, by reflecting the information-flow

meaning behind the abstract list of access modes. When there is a close match between the model's access modes and those enforced in the hardware for memory access, e.g., read and write, finding the proper interpretation is easy. Otherwise, one has less assurance that the model is implemented accurately.

Real security policies are not pure. The *-property in the Bell-LaPadula model, for example, has an exception built into it for trusted subjects. How does one decide whether a particular subject deserves to be trusted? One cannot really answer this in the context of the model, though we have noted that it is possible to use additional structure in the model to limit the privilege of a trusted subject, through sharing control with other subjects or adding type-enforcement restrictions.

It was a pleasant discovery of Biba's that a limited form of nondiscretionary integrity control is possible simply by reinterpreting the meaning of labels. Modern systems should be designed to be flexible enough to take advantage of the strict integrity trick, despite the fact that it does not address the prior question of how to qualify subjects for high integrity, nor does it implement the type-enforcement or pipeline policies called for in commercial applications.

It is difficult to tell how best to use MAC system models for database systems or networks. There are two levels at which the MAC approach can be used. One level is at an external interface, where objects are complex abstractions such as relations, views, virtual connections, or datagrams. This is the most natural level at which to describe the system security policy as it is visible to users, but so much software is used to support it that it is difficult to assure correct implementation. The other level is at the interface to the underlying secure operating system kernel, if there is one, where objects are segments of memory. This is the level at which access control is enforced, and where one has the most assurance that a simple information-flow policy is implemented. Both levels seems to be needed.

All access-control models have the failing that they assume that information flow can occur only when an appropriate access mode has been granted. In fact, information is communicated by all kinds of events, including the refusal of access, leading to covert channels. It is a tribute to the perseverance of researchers that they not only understand how this is possible, but they have developed proof techniques for finding covert channels that can be used in practice, albeit with some difficulty at present.

The developments in information-flow modelling are exciting because they are still evolving in a clear direction. Starting with the underlying notion of information flow as an inference about the possible values of a sensitive data source, leading to the non-interference concept in deterministic machines, the

following advances have been made:

- An equivalent state-transition formulation (unwinding).
- A technique for detecting covert channels based on the state-transition version.
- A stronger but composable definition for nondeterministic systems (restrictiveness).

Restrictiveness is not the final answer, because it has not been shown to be the weakest definition that still guarantees composability and which reverts to non-interference on deterministic systems. Analysis techniques based on information-flow approaches also need to be developed further. Perhaps, one day, the present dichotomy between access-control policy and covert-channel analysis will disappear, and the two will be subsumed in theory and practice under a single methodology.

References

Anderson, J. P. (1972). "Computer Security Technology Planning Study," Vol. 1. ESD-TR-73-51, AD 758 206. James P. Anderson and Co., Fort Washington, Pennsylvania.

Bell, D. E. (1973). "Secure Computer Systems: A Refinement of the Mathematical Model." ESD-TR-73-278, Vol. III. The MITRE Corporation, Bedford, Massachusetts.

Bell, D. E. (1988). Concerning "Modelling" of Computer Security. *Proc. 1988 IEEE Symp. Security and Privacy*, pp. 8–13.

Bell, D. E., and LaPadula, L. J. (1973). "Secure Computer Systems: Mathematical Foundations." ESD-TR-73-278, Vol. I. The MITRE Corporation, Bedford, Massachusetts.

Bell, D. E., and LaPadula, L. J. (1975). "Secure Computer System: Unified Exposition and Multics Interpretation." ESD-TR-75-306. The MITRE Corporation, Bedford, Massachusetts.

Benzel, T. C. V. (1984). Analysis of a Kernel Verification. *Proc. 1984 IEEE Symp. Security and Privacy*, pp. 125–133.

Biba, K. J. (1977). "Integrity Considerations for Secure Computer Systems." ESD-TR-76-372. The MITRE Corporation, Bedford, Massachusetts.

Boebert, W., and Kain, R. (1985). A Practical Alternative to Hierarchical Integrity Policies. *Proc. 8th National Computer Security Conf.*, pp. 18–27.

Burke, E. L. (1974). "Synthesis of a Software Security System." MTP-154. The MITRE Corporation, Bedford, Massachusetts.

Casey, T. A., Vinter, S. T., Weber, D. G., and Varadarajan, R. (1988). A Secure Distributed Operating System. *Proc. 1988 IEEE Symp. Security and Privacy*, pp. 27–38.

Cheheyl, M. H., Gasser, M., Huff, G. A., and Millen, J. K. (1981). Verifying Security. *ACM Computing Surveys* **13** (3), 279–339.

Clark, D. D., and Wilson, D. R. (1987). A Comparison of Commercial and Military Security Policies. *Proc. 1987 IEEE Symp. Security and Privacy*, pp. 184–189.

Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. *Commun. ACM* **13** (6), 377–387.

Cohen, E. (1978). Information Transmission in Sequential Programs. *In* "Foundations of Secure Computation" (R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton, eds.), pp. 297–336. Academic Press, New York.

Denning, D. E. (1976). A Lattice Model of Secure Information Flow. *Comm. ACM* **19** (5), 236–242.

Denning, D. E., and Denning, P. J. (1977). Certification of Programs for Secure Information Flow. *Comm. ACM* **20** (7), 504–513.

Denning, D. E., Akl, S. G., Heckman, M., Lunt, T. F., Morgenstern, M., Neumann, P. G., and Schell, R. R. (1987a). Views for Multilevel Database Security. *IEEE Trans. Software Eng.* **SE-13** (2), 129–140.

Denning, D. E., Lunt, T. F., Schell, R. R., Heckman, M., and Shockley, W. (1987b). A Multilevel Relational Data Model. *Proc. 1987 IEEE Symp. Security and Privacy*, pp. 220–233.

Feiertag, R. J., Levitt, K. N., and Robinson, L. (1977). Proving Multilevel Security of a System Design. *Proc. 6th ACM Symp. Operating System Principles*, pp. 57–65.

Fraim, L. J. (1983). SCOMP: A Solution to the Multilevel Security Problem. *IEEE Computer Magazine* (July, 1983), 26–34.

Glasgow, J. I., and McEwen, G. H. (1987). The Development and Proof of a Formal Specification for a Multi-level Secure System. *ACM Trans. Computing Systems* **5** (2), 151–184.

Goguen, J. A., and Meseguer, J. (1982). Security Policies and Security Models. *Proc. 1982 IEEE Symp. Security and Privacy*, pp. 11–22.

Goguen, J. A., and Meseguer, J. (1984). Unwinding and Inference Control. *Proc. 1984 IEEE Symp. Security and Privacy*, pp. 75–85.

Graham, G. S., and Denning, P. J. (1972). Protection—Principles and Practice. *Proc. IFIPS Sprint Joint Computer Conf.*, pp. 417–479.

Graubart, R. D., and Woodward, J. P. L. (1982). "A Preliminary Naval Surveillance DBMS Security Model." MTR-8475. The MITRE Corporation, Bedford, Massachusetts.

Grohn, M. J. (1976). "A Model of a Protected Data Management System." I. P. Sharp Associates Limited, Ottawa, Canada.

Guttman, J. D. (1987). Information Flow and Invariance. *Proc. 1987 IEEE Symp. Security and Privacy*, pp. 67–73.

Haigh, J. T., Kemmerer, R. A., McHugh, J., and Young, W. D. (1986). An Experience Using Two Covert Channel Analysis Techniques on a Real System Design. *Proc. 1986 IEEE Symp. Security and Privacy*, pp. 14–24.

Harrison, M. A. (1985). Theoretical Issues Concerning Protection in Operating Systems. *In* "Advances in Computers," Vol. 24 (M. C. Yovits, ed.), pp. 61–100. Academic Press, New York.

Harrison, M. A., Ruzzo, W. L., and Ullman, J. D. (1976). Protection in Operating Systems. *Comm. ACM* **19** (8), 461–471.

Hinke, T. H., and Schaefer, M. (1975). "Secure Data Management System." RADC-TR-75-266. System Development Corporation, Santa Monica, California.

Jones, A. K., and Lipton, R. J. (1975). The Enforcement of Security Policies for Computation. *ACM Operating Systems Rev.* **9** (5), 197–206. (Also in *J. Comput. Syst. Sci.* **17**, 35–55.)

Lampson, B. W. (1971). Protection. *Proc. 5th Princeton Conf. Inf. Sci. Syst.*, pp. 437–443.

Lampson, B. W. (1973). A Note on the Confinement Problem. *Comm. ACM* **16** (10), 613–615.

Landwehr, C. E. (1981). Formal Models for Computer Security. *ACM Computing Surveys* **13** (3), 247–278.

LaPadula, L. J., and Bell, D. E. (1973). "Secure Computer Systems: A Mathematical Model." ESD-TR-73-278, Vol. II. The MITRE Corporation, Bedford, Massachusetts.

Lee, T. M. P. (1988). Using Mandatory Integrity to Enforce "Commercial" Security. *Proc. 1988 IEEE Symp. Security and Privacy*, pp. 140–146.

Lipner, S. B. (1982). Non-Discretionary Controls for Commercial Applications. *Proc. 1982 IEEE Symp. Security and Privacy*, pp. 2–10.

McCullough, D. (1987). Specifications for Multi-level Security and a Hook-Up Property. *Proc. 1987 IEEE Symp. Security and Privacy*, pp. 161–166.

McCullough, D. (1988a). "The Theory of Security in Ulysses." Odyssey Research Associates, Ithaca, New York.

McCullough, D. (1988b). Noninterference and the Composability of Security Properties. *Proc. 1988 IEEE Symp. Security and Privacy*, pp. 177–186.

McHugh, J., and Moore, A. P. (1986). A Security Policy and Formal Top Level Specification for a Multi-Level Secure Local Area Network. *Proc. 1986 IEEE Symp. Security and Privacy*, pp. 34–39.

McLean, J. (1987). Reasoning About Security Models. *Proc. 1987 IEEE Symp. Security and Privacy*, pp. 123–131.

McLean (1988). The Algebra of Security. *Proc. 1988 IEEE Symp. Security and Privacy*, pp. 2–7.

Millen, J. K. (1978). Constraints, Part II. Constraints and Multilevel Security. *In* "Foundations of Secure Computation" (R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton, eds.), pp. 205–222. Academic Press, New York.

Millen, J. K. (1984). A1 Policy Modelling. *Proc. 7th DoD/NBS Computer Security Conf.*, pp. 137–145.

Millen, J. K., and Cerniglia, C. M. (1984). "Computer Security Models." MTR-9531. AD A 166 920. The MITRE Corporation, Bedford, Massachusetts.

National Computer Security Center (1985). Department of Defense Trusted Computer System Evaluation Criteria. DoD 5200.28-STD.

Neumann, P. G., Boyer, R. S. Feiertag, R. J., Levitt, K. N., and Robinson, L. (1977). "A Provably Secure Operating System: The System, its Applications, and Proofs." Stanford Research Institute, Menlo Park, California.

Parnas, D. L. (1972). A Technique for Software Module Specification With Examples. *Comm. ACM* **15** (5), 330–336.

Rushby, J. (1985). The SRI Security Model. Computer Science Laboratory, SRI International.

Walter, K. G., Ogden, W. F., Rounds, W. C., Bradshaw, F. T., Ames, S. R., and Shumway, D. G. (1974a). "Primitive Methods for Computer Security." Department of Computing and Information Sciences, Case Western Reserve University, Cleveland, Ohio.

Walter, K. G., Ogden, W. F., Rounds, W. C., Bradshaw, F. T., Ames, S. R., Biba, K. J., Gilligan, J. M., Schaeffer, D. D., Schaen, S. I., and Shumway, D. G. (1974b). "Modeling the Security Interface." Department of Computing and Information Sciences, Case Western Reserve University, Cleveland, Ohio.

This Page Intentionally Left Blank

Evaluation, Description and Invention: Paradigms for Human-Computer Interaction

JOHN M. CARROLL

User Interface Institute
IBM T.J. Watson Research Center
Yorktown Heights, New York

1. Introduction

A vivid image of the recent evolution of computer technology is that of a "race" between function and usability. New technologies and new capabilities become available to users faster than user problems can be studied, understood and addressed. For example, the many user studies of word-processing applications carried out over the past decade focused their attention on keyboard-oriented, stand-alone systems with small and low-resolution monochrome displays. In 1981, our group at the Watson Research Center turned attention to secretaries learning to use such word-processing applications. At the time, this was a novel application; computer editing was still largely the province of programmers revising code.

But now, and without a finished analysis of word processing, the frontier of usability has been pressed onward by the development and introduction of new applications and new interface technologies. Communication applications such as electronic mail and computer conference support raise usability challenges far more diverse than those raised by the extension of

Copyright © 1989 by Academic Press, Inc.
All rights of reproduction in any form reserved.
ISBN 0-12-012129-8

word processing to nonprogrammers. In the current technology, multiple users cooperatively access multiple applications via an extremely heterogeneous collection of workstation types. And even as the usability issues in these new domains are being articulated and explored, leading-edge prototypes are introducing gestural (e.g., handwriting) and speech input and interactive video output. Such new developments are occurring more rapidly, more broadly across the industry, and affecting more users all the time.

The race between function and usability has made the area of human-computer interaction (or HCI) a very high-profile research area within computer science and within the computer industry: it is difficult to develop usability science and technology fast enough, but it is also critical to do so. Indeed, the race has created the need for chapters such as this one. However, this attention has also helped to expose some fundamental perplexity about what the field is and how it is supposed to work. It is still the case that HCI research has its principal effect on *discussions* of usability and user-interface design and only a small, derived effect on actual *practice* in the design and development of computer systems and applications.

What is the goal of HCI research? There need not be a single answer to this question. But the more answers there are, and the more irreconcilable the various answers are, the more fragmented the field will appear. In HCI there are many answers to this question. One traditional answer comes from the field of Human Factors: HCI needs to provide methods and metrics for evaluating the usability of computers. A second answer comes from Cognitive Science: HCI is a testbed for the application of cognitive psychology to a real problem domain. A third answer comes from the exigencies of the computing industry: HCI must help guide the definition, invention and introduction of new computing tools and environments.

The practice of HCI is even more fragmented than its goals might imply. For example, some varieties of human factors evaluation explicitly suggest that developing cognitive science theories of HCI may *impair* progress in understanding usability (Whiteside and Wixon, 1987). On the other hand, Newell and Card (1985) warn that psychology might be driven out of HCI by computer science unless it can develop predictive cognitive models, coining the slogan "hard science drives out the soft." Yet even the most developed cognitive models in HCI have had no significant impact on the design of user interfaces (Carroll and Campbell, 1986). Moreover, it is paradoxically true that product innovations in user-interface design have generally *led* HCI research rather than following from it in the conventionally assumed flow of "technology transfer" from Research to Development. The recent impact of the Apple Macintosh illustrates this.

Perhaps these conflicting and fragmented views of HCI can be understood as consequences of the race between function and usability, of the rapid growth in needs, activities and expectations. Perhaps the current perplexity

about HCI reflects an intermediate state in a true evolution toward more effective approaches to understanding the usability of computer systems and applications. In this chapter I take such an historical view, identifying three distinct paradigms, or orientations, to HCI research and application. Initially, HCI work focussed on empirical laboratory *evaluation* of computer systems and techniques. Subsequently, empirical studies of usability were organized by and addressed to cognitive theoretical *description* of human behavior and experience. Currently, the focus of HCI work is shifting toward a more directive role in *invention*, design and development. The progression of these three paradigms comprises a case study of a field discovering what it is about, and, more generally, of the variety of roles available in the psychology of technology.

2. Human Factors Evaluation

The traditional role of psychologists working in the context of computer applications and services is empirical evaluation of usability. The original research arena of human-computer interaction is the psychology of programming and the professional programmer (Curtis, 1985; Shneiderman, 1980). A prototypical example of this paradigm is a set of experiments conducted by Sheppard *et al.* (1979). In one of these, participants were given 20 minutes to reconstruct from memory a Fortran program of 26–57 lines that they had studied for the preceding 25 minutes. Two approaches to "structured" program organization (Dijkstra, 1972) were contrasted with a "convoluted" organization (including backward exits from DO loops, arithmetic IFs, and unrestricted GOTOs). Reconstructive memory for the convoluted program organization was poorer (i.e., error rates were higher) than for either of the structured organizations (though only in one case was the difference statistically significant).

Such early work in the human factors of programming was important in demonstrating the feasibility of empirical assessment. By addressing some of the timely issues of the day, it broadened the grounds of debate in software technology from formal analysis and system performance to include usability and productivity issues. The basic paradigm of directly comparing two alternate designs in a usability evaluation is still the standard of practice in much HCI research and in many product development laboratories.

2.1 Direct Empirical Contrast

The development of empirical methodologies for evaluation, and the exercise of these methodologies in the context of software and system design, is a continuing need in HCI. Direct empirical measurement is still the only

adequate means of assessing the usability of software techniques and computing artifacts (Carroll and Rosson, 1985; Curtis, 1980; Gould and Lewis, 1985). Establishing the importance of usability to the success of computing systems and techniques, and developing and promoting empirical methodologies to make usability evaluations have been major foci of HCI work.

From the start, HCI evaluation studies were strongly influenced by research practice in experimental psychology: emphasis was placed on tightly controlled laboratory approaches. From an historical standpoint, this was a reasonable move: there was an acute lack of theory and methodology for investigating usability. These laboratory studies generally took the form of direct contrasts: computing artifacts or techniques were directly pitted against one another in a brief but behavior-intensive measurement session. This evaluation work produced a variety of findings, often framed as guidelines for software-development practice and user-interface design, generally of the form "A is better than B." And perhaps even more importantly, the work set a more objective standard for usability evaluations, and provided a systematic basis for scrutinizing designers' hopeful intentions and trade-press reviewers' glib comments.

However, there are many limitations inherent in the laboratory-based direct-contrast methodologies of experimental psychology. These limitations become clear when the methodologies were applied in the complex practical contexts of HCI design. Controlled laboratory studies of software are difficult to design and carry out. The investigator needs to master programming languages and computer applications in order to be in a position to assess others' performance and to interpret their experiences. The experimental tasks that are studied necessarily require skilled human participants and involve learning and using very complex tools. This is expensive and time-consuming research. Such difficulties just don't come up when one takes an experimental approach to memorizing nonsense syllables, the stock-in-trade of traditional experimental psychology, or to making timed responses to meaningful but simple objects such as isolated words, its more modern variant.

In experimental psychology, the sheer differences in recall rate or response times may be all there is to know about a person's performance in a task: the situations are relatively simple. Understandably perhaps, such work is directed at collecting straightforward quantitative indicators of performance such as task times and error rates, and formally testing these for statistical significance of direct contrasts (that is, computing the probability that obtained score differences might have occurred by chance). HCI situations, however, are not simple at all. In many cases it may be more important to know how people approach a task, or how they feel about their performance, than it is to know how quickly or successfully they perform. Nevertheless, the early commitment of HCI evaluation work to direct-contrast studies created a

strong bias for collecting quantitative indicators of performance, such as time and success measures, and against placing primary, or even equal emphasis on qualitative data (which in other human factors contexts have often played a more prominent role; Chapanis, 1959, pp. 23–95).

These constraints of direct-contrast laboratory methods took a toll on the relevance of HCI evaluation work. The difficulties of designing and conducting controlled experiments in complex circumstances inclined investigators to make use of scaled-down tasks such as, for example, memorization and reconstruction of small programs. The focus on quantitative differences inclined investigators to focus on the simplest of performance measures. This undermined the fundamental objectives of human factors evaluation, transforming questions about complex human behavior and experience in complex computing environments into simple scores of performance on toy-scale tasks. Such work could not answer the underlying "why" questions that motivated human factors evaluation in the first place; it could not provide the depth of understanding necessary to help guide the design of new software techniques and applications.

Yet this style of work became quite pervasive. Ledgard *et al.* (1980) assessed the use of symbolic notations in text-editor commands by contrasting a command language having extremely complicated symbolic conventions with one almost free of these. Murrel (1983) contrasted message-based and window-based communication for a cooperative decision-making task. Holt *et al.* (1987) contrasted object-oriented design with more standard approaches. But exactly what is it about symbolic notations that is bad? What is it about window-based communication and object-oriented design that is good? None of these projects resolved the overall evaluation issue it posed. And none collected detailed enough information to contribute to a conceptual understanding of the issues involved.

Worst of all perhaps, these simplications frequently did not even produce the statistically significant differences they were adopted to facilitate. The use of indentation to highlight structure in program listings seems intuitively like a good idea. It's a simple factor that can in principle be conveniently removed from the complications of the real programming process for direct-contrast laboratory study. However, Love (1977), Shneiderman and McKay (1976) and Weissman (1974) all failed to find significant benefits of indentation. Studies of variable names have produced a conflicting potpourri of results; sometimes mnemonic names are more effective than non-mnemonic names and sometimes not (Schneiderman, 1980, pp. 70–71). The daunting possibility remains that it was *because* of the trivial tasks that were studied and the limited types of data that were collected and analyzed that no differential benefits were found.

Such practical problems with direct contrasts encouraged experimental designs contrasting extreme positions, again to increase the possibility of

measuring statistically significant differences. The assessment of symbolic conventions by Ledgard *et al* (1980) contrasted extremely complicated examples of such conventions with an extreme absence of them. Liebelt *et al.* (1982) showed that a menu system was easier to learn when the menu hierarchy was organized than when it was disorganized(!). Indeed, in the Sheppard *et al.* (1979) experiment, several alternate approaches to "structured" programming were consistently indistinguishable based on the data, but the extreme alternative of "convoluted" programming produced significantly poorer performance than either of the structured approaches. In a sense, this study did not so much verify the benefits of deliberately structuring code as it did the risks of deliberately mis-structuring it. (Obvious and extreme evaluation contrast are still sometimes professionally encouraged as long as they employ "an interesting methodology" (Green, 1987, pg. 6).)

Finally, human factors evaluation work is highly constrained by the often prodigious amounts of time required to make direct experimental contrasts of alternatives. Indeed, it seems logically doomed to consume more time than the evolution of software it is intended to guide. By the time the Sheppard *et al.* (1979) paper appeared, structured programming methods were already the established practice. The evaluation work confirmed what had already happened, rather than playing a causal role in the evolution of practice. This limitation of the evaluation paradigm for HCI could be called the "evaluation dilemma": one cannot evaluate something that does not yet exist, hence direct evaluation always lags development by some fraction of a development cycle (Carroll, 1987a).

In sum, the exigencies of direct-contrast laboratory work entailed compromises in the face validity of the work itself, and, in the end, often failed to produce definitive or timely evaluations. How should programs be structured? How should hypertextual information systems be navigated? One cannot answer these questions with a few simple performance measures, but they are surely *empirical* questions. Answering them would involve developing a detailed understanding of what people do and try to do with programs and applications and the rich interaction of these goals and actions with the constructs of programming languages, the facilities of computing environments, aspects of the workplace, and many other factors.

These complexities have had a predictable effect: even in quarters where human factors evaluation is the official operating paradigm, most of the impact of psychology on the development of technology has come about through task analysis or consulting. Indeed, to a considerable extent human factors evaluation has become an historical stage in the development of current HCI. We return to the curious schism between what is officially anointed as standard practice and what is in fact the standard practice in later discussion of the invention paradigm for HCI.

2.2 Lack of Theory

The guiding hope in doing evaluation work is that the data collected and the methods developed can cumulate into coherent analyses about *why* some systems and techniques are more usable than others, and about *how* to enhance the usability of future systems and techniques. It is a bottom-up approach to developing theory. However, directly contrasting two complex situations (e.g., two versions of a system) to determine which one is better is a poor vehicle for sorting out and saving experience. Complex alternatives with no *a priori* theoretical analysis do not become interpretable merely by virtue of a simple horse race. It would take an infinity of such "one-off" contrasts to build a theory from the bottom up. Even the simple and controlled situations studied in experimental psychology would be intractably indeterminate without top-down theoretical direction.

Many of the difficulties with direct-contrast evaluations can be attributed to this lack of theory. The use of toy-scale problem domains and simple, quantitative measures is problematic in that without a theory of HCI domains there is no way to know whether a toy problem is representative of a real problem or not. There is no way to know whether one is studying a coherent part of the real problem, or an accidental and idiosyncratic case. Can an analysis of writing 50-line programs be scaled up to the problem of writing 5000-line programs? Is the task of pointing a cursor at an arbitrary screen location a coherent part of the task of pointing a cursor in the course of editing text? Are interpretations of isolated system events related to interpretations of the very same events embedded in a real stream of user interaction? Answering such questions is impossible without a theory with which to interpret the toy situations and to extrapolate from them to real situations.

Sheil (1981), for example, noted that complexity is not linear with program length. It certainly seems that the task of editing a 5000-line program raises problems of navigation and naming conventions that are just not raised in the task of editing a 50-line program. Elements of HCI situations may interact and trade off in different ways as the problem scale or the task changes. Is avoiding GOTO statements more or less important than employing indentation in a program listing? And are there contexts in which the relation is inverted? Again, without a theory there is no way to extrapolate these interactions. Indeed one can do little more than organize separate studies on the basis of superficial features (e.g., as pertaining to variable names or menu systems). Without a theory of, for example, how people understand, name, and remember entities, there is no way to work back from a variety of performance differences obtained in a variety of experimental settings to an explanation of the underlying concepts that caused the differences (see Newell, 1973).

In the absence of a theoretical framework for understanding usability, HCI

evaluation work has had to address issues at a very large grain of analysis. Hauptmann and Green (1983), for example, contrasted a natural-language interface with a menu interface for creating business graphics (failing to find any significant differences in time, errors or attitudes). Of course, contrasting natural language with menus is painting with a rather broad stroke: how could a single experimental contrast resolve such a multifaceted contrast? Were the two interfaces individually optimized to be the best interface possible in their respective interface styles? Were they controlled to have the same functional capabilities and the same task-relative functional capabilities? The same kinds of questions arise for the examples discussed earlier, evaluating structured programming, object oriented programming and symbolic notations. The lack of theory forces these crude contrasts; but the crude contrasts prohibit pertinent or univocal results.

Methods and theories in software technology are often collections of loosely connected prescriptions. Ideas such as structured programming and direct manipulation (Shneiderman, 1983) are important theoretical concepts, and they surely carry empirical consequences. But they are not falsifiable in the Popperian sense (Popper, 1965): one cannot hope to reject such ideas *tout court* on the basis of isolated laboratory tests; to try to do so is to get the logic of the inquiry wrong. From our current perspective of a few years hence, it is clear that no outcome of the Sheppard *et al.* (1979) study could have rejected structured programming as an appropriate prescriptive theory. The real evaluation need is for detailed qualitative information that can guide the revision and integration of such ideas. The issue is not whether structured programming is good, or indeed whether it is better than some other approach; the issue is what structured programming really consists of, how in detail it affects actual programming tasks, and how it can be integrated into routine programming practice.

The assessment goal is just too limiting: a paradigm that merely evaluates distinctions articulated by others deprives itself of playing any directive role (Sheil, 1981). In this context, we can understand why studies such as Sheppard *et al.* (1979) failed to lead to the development of an articulated theory of programming: the evaluation enterprise bound itself to what already existed, commenting at a high level on the appropriateness of specific techniques from the mid 1970s. A poignant example is the work showing that input error rates are reduced when using teletype terminals instead of visual display units (Walther and O'Neil, 1974; Carlisle, 1970). It was never a possibility that teletype terminals would supplant visual display units through the course of technological evolution, quite the contrary. The bald evaluation result, without specific implications for the design of future visual display devices, can only be seen as an historical curiosity.

Empirical evaluation of software and systems is a key to usability. But it is a

separate question whether a *science* of human-computer interaction can arise out of this activity. In fact, it did not. The evaluation paradigm introduced psychology and psychologists to the HCI problem domain. It was a platform for establishing the importance of usability and for developing empirical approaches to measuring the usability of systems and software. However, its methodological commitments and lack of theory cast it in a supporting role in emerging software and user-interface science: more of a commentator on new technology than a directive force. The challenge that this raised was how psychology could play a more directive role in the development of new software and user-interface technology.

3. Cognitive Description

In the early 1980s there was a shift toward bringing HCI research under the aegis of broader psychological theory. Shneiderman (1980, pg. 51), for example, used the classic paper of Miller (1956) on human information processing limitations to derive the prescription that programmers avoid the use of GOTO constructs. Shneiderman analyzed the process of understanding programs as involving the recoding of lines of code into meaningful "chunks." GOTO jumps in a program text disrupt this structure by functionally chunking nonadjacent lines of code. Card *et al.* (1983) published a compelling monograph adapting information-processing psychology to the description of fluent user interaction with text editors. These efforts had an enormous effect, enlarging and intensifying interest in the psychology of usability both within computer science *and* within psychology.

This shift confronted one of the key limitations of earlier work, the lack of theory. Tying specific empirical results to theories of human information processing provided means to integrate diverse results, to resolve nonsignificant or conflicting findings, to dampen the distortions of poor research, but most importantly to develop abstractions that, in principle, could help lead the development of software technology and user-interface design.

However, this work also raised new issues and problems. Aligning HCI phenomena with cognitive descriptions of those phenomena is useful to the extent that the cognitive descriptions themselves are rich, revealing and well-integrated. In fact, psychological theory is at least as fragmented as software theory and methodology. Building a psychology of usability by placing this body of fragmented theory into correspondence with software situations risks inheriting the fissures as well as the solid ground. Ironically, cognitive description work also threatened the major achievement of human factors evaluation, namely, establishing the centrality of direct usability testing to the ultimate success of computing systems and techniques. The cognitive description paradigm entrained a strongly analytic conception of software

design, raising the question of how much direct evaluation might be necessary if a good theory were in hand.

3.1 Breadth versus Depth

Scientific psychology seeks to understand behavior and experience by providing laws, concepts, and explanations. However, there are severe limits on what types of phenomena psychology can address with these goals and tools; there are ranges over which the goals and tools make sense and outside of which they do not. In particular, academic psychology typically attempts to capture generalizations across domains. But fine details of specific task situations can be very important: what a person thinks and decides to do is often ascribable to knowledge of a single fact, e.g., the name of a particular command in a particular system. These fine-grained details serve as boundary markers for theorizing: scientific laws that must refer to individual facts as conditions seem unwieldy, and psychologists routinely make a strategic retreat to abstract or artificial domains to control such details.

This is a reasonable heuristic, with extensive precedent in the sciences. Classical point-mass mechanics is developed under the idealization of frictionless contact, even though there are no frictionless systems. Other theoretical apparatus has been developed to add back the effects of friction in real systems. The difficult details of friction are treated as "perturbations" of the classical theory (Gleick, 1987). Similarly, the traditional research strategy in psychology has been to focus on sweepingly general issues and distinctions under the idealization that domain and situation context can be ignored. Basic psychological research addresses topics such as the "structure of memory," but not, for example, "memory for Unix commands" (Norman, 1981). It tries to resolve "big" issues such as "is there a separate mental type for imagery?" (Pylyshyn, 1973; Paivio, 1971).

It turns out that describing frictionless contact provides a useful foundation for understanding the motion of real objects in real circumstances. Even though the effects of friction are not simple, treating these effects as perturbations of an idealized theory has also proven tractable in engineering applications (for example, computing trajectories). The question is whether the same basic strategy is useful in psychology. This is an open question. Newell (1973), for example, criticized the pursuit of sweeping dichotomies such as existence of a separate mental type for imagery, saying "you can't play twenty questions with nature and win." Indeed, the emergence in the 1980s of Cognitive Science as a broader discipline, incorporating psychology with the serious consideration of the structure of task domains, can be seen as a response to traditional idealizations (Carroll, 1988).

Chase and Simon's (1973) classic study of expertise in chess showed that, for

a reconstructive memory task, chess masters tended to recall piece positions in attack and defense groupings. This study has had two very different legacies. On the one hand, it opened up a variety of questions about domains. How are chess piece groupings indexed in a player's memory; how they are accessed in realistic tasks (such as playing chess, as opposed to reconstructive memory for arbitrary board positions); how does expertise in chess develop through significant spans of time? Many of these issues have been pursued and in a variety of domains (see Chi et al., 1988), though many would argue that the work still takes too narrow a view of the process of attaining expertise and of the nature of expert knowledge and performance (e.g., Dreyfus and Dreyfus, 1986).

On the other hand, Chase and Simon's result was sweepingly generalized as "experts have chunks," and has been mechanically replicated in domain after domain. There is no rich and well-integrated theory of either experts or chunks outside of considerations of specific domains. Thus, these studies show only that when humans know something about a domain and are asked to do reconstructive memory tasks of an arbitrary sort, they use what they know to do the task. A series of these studies have been undertaken in HCI contrasting memory performance for scrambled and unscrambled program listings (Adelson, 1981; McKeithen *et al.*, 1981; Shneiderman, 1980). This work showed that people with programming experience can use knowledge of language structures in organizing their memories.

This finding has not led to rich understandings of how people achieve expertise in programming or about how programming knowledge is indexed in memory and accessed in performance. It has not helped to guide the development of new software tools and environments. These cognitive descriptions do not address and provide no guidance in practical aspects of programming (the design of programming languages, environments, education, etc.); they do not even engage issues specific to the domain of programming (the types of modules one would want in a library to facilitate code reusability).

An extensive tradition of psychological research describes learning, memory and error patterns for paired-associates, the classic nonsense syllable (e.g., Esper, 1925; Postman and Stark, 1962). This work has been applied to the analysis of user performance with various types of command languages (Barnard *et al.*, 1981; Carroll, 1982; Landauer *et al.*, 1983). For the most part, these applications have been no less mechanical than those of the "experts have chunks" work. Yet they have been relatively more successful in that the cognitive descriptions developed for command language interactions have had fairly specific prescriptive content for command language design. Indeed, HCI research on command names has led to specific revisions in philosophical and linguistic conceptions about what names are (Carroll, 1985).

But this work, and indeed all cognitive description work in HCI, is subject to a very fundamental problem in the underlying logic of the inquiry. Psychology concerns itself with *existence*: is there a separate mental type for imagery? HCI, like any applied science domain, concerns itself with *impact*: how much of a difference will certain types of consistency make in the learnability of a command language? This is why the "experts have chunks" work seems reasonable from the perspective of our curiosity about chess masters and other experts, but difficult to apply in the face of questions about how to support experts and facilitate the development of expertise. This is also why the use of extreme contrasts, such as scrambled programs versus structured programs, can make sense in the pursuit of basic theory, but much less so in the pursuit of meaningful application.

Landauer (1987a) has recently called attention to this in observing that while basic psychology routinely focusses on the "significance" of effects, it typically disregards the *size* of effects. Cognitive descriptions framed in terms of existence dichotomies can be assessed by the statistical significance of direct contrasts: do expert programmers chunk more than novices? However, such differences do not guarantee that the effects will be large enough to matter. Would it matter if experts reliably chunked 2% more than novices? Would it matter if scrupulously consistent command languages were learned 3% faster than randomly consistent languages? To determine the practical size of effects one needs to consider cost-benefit tradeoffs in realistic tasks. Chunking may have a big effect on people trying to memorize scrambled little programs, but the size-of-effect question forces attention to real programmers writing and reading real programs. The two situations might be quite different.

3.2 Design by Deduction

HCI is fundamentally a design domain: it exists in the first place because of the need to design more usable computing artifacts for people to use. Design in a complex and poorly charted domain can seem like trial and error. How should user-interface design work proceed to ensure more usable user interfaces? The human factors evaluation paradigm sought to address this kind of question by providing methodology for directly evaluating design techniques (such as structured programming) and particular artifacts (for example, a particular programming language or programming environment). But direct evaluation operates on a case-by-case basis. The cognitive description paradigm sought to improve upon this by providing theoretical abstractions beyond the specific cases (see Moran, 1981).

Card *et al.* (1983) made what is surely the most thorough and disciplined attempt to interpret and develop modern information-processing psychology into a foundation for the design of computer systems. In their GOMS model

(an acronym for Goals, Operators, Methods and Selection rules), users hierarchically decompose their goals into successively finer subgoals until these match a basic set of methods. The user has rules for selecting methods appropriate to the current situation, and each method itself consists of a sequence of operators, keypresses and hand motions. This analysis was fitted to a variety of text-editing performance data, in many cases yielding consistent values for the model's parameters.

However, the theory proved quite limited in application to user-interface design. GOMS was not able to describe problem-solving activity, only routine, over-practiced performance. In fact, it could not describe errors at all, even though nearly a third of the routine behavior it sought to describe consisted of error and error recovery. It was also severely hampered by the race between function and usability: by the time it had produced good performance descriptions for error-free, over-practiced behavior on line-oriented editors, the focus of concern in user interfaces and end-user applications had moved on to other problem areas. (See Carroll and Campbell (1986) for further discussion.) The work had its greatest impact on relatively low-level aspects of human-computer interaction, such as the analysis of pointing devices (Card *et al.*, 1978). Indeed, it appears that this approach may only work for user-interaction events on the order of one second in duration in which errors are extremely rare and/or extremely regular(!), and for technological contexts that are unchanging on the order of decades (Newell and Card, 1985). Few design problems in HCI fall into this rather severe category.

Most cognitive description work is far less theoretically ambitious than the GOMS work. For example, the use of menu selection as an alternative to typed commands is sometimes "deduced" from the fact that humans are better at recognition than at recall (e.g., Tennant, *et al.*, 1983). This is terribly oversimplified. Users of menu systems must deal with formidable navigation problems (MacGregor and Lee, 1987; Robertson *et al.*, 1981). They must deal with complex morphological, semantic and referential relations *between* various selection names (Carroll, 1985). Here again, the evolution of user-interface technology is complicating the simple dichotomies: rich aliasing (Gomez and Lochbaum, 1985) may substantially mitigate the relative difficulty of recall, and alternative approaches to menu design may carry differing performance implications (pop-up menus, multiple-selection menus, active forms). Finally, though the advantage of recognition over recall is an established sweeping principle in psychology (e.g., Crowder, 1976), Black and Sebrechts (1981) have observed that there are circumstances in which the reverse is true.

We earlier considered Shneiderman's (1980) reference to Miller's (1956) analysis of human information-processing limitations in grounding the prescription to avoid GOTOs. Miller's specific argument, however, does not

consider spatial or temporal proximity of items to be "chunked." Accordingly, the GOTO prescription cannot be deduced from Miller's analysis. Indeed, virtually nothing of much interest could be *deduced* from the specifics of Miller's analysis. The connection is more informal: Miller's work called attention to the (obvious) fact that humans are limited with respect to the information they can manage; Shneiderman was inspired by this to suggest a particular tactic for easing information management in programming. The informality of the theoretical linkages is not specially problematic: the non-psychological-theory components of HCI do no better (e.g., what is an interface toolkit?). Having theories cogent enough and pertinent enough to even informally direct and inspire design work is a big advantage.

The problem *vis-a-vis* design by deduction is that in none of these examples of cognitive description applied to design do we have in hand the ancillary theoretical apparatus to deductively bridge between the "leading claims" and the implementation details. GOMS *is* probably a reasonable first approximation framework for thinking about task analysis. Recognition probably *is* easier than recall in many circumstances. GOTOs probably *do* strain human information-processing capacity. But to use this theoretical material deductively in design we need to know precisely how the details of given situations interact with and modulate the psychological principles. None of the theories is complete enough to tell us this. Hence none can be used deductively.

To an extent, this lack can be addressed through theory development. For example, Polson (1987) has developed the GOMS approach into a potentially more useful design tool. However, other considerations indicate that HCI design can never be rendered deductive. The particular complexity of software technology stems from the fact that everything inherently interacts with everything else (Brooks, 1987). The technological context plays an important role in determining whether an idea will survive at all. For example, object-oriented techniques have been seen as a major advance in software technology, but the successful use of these techniques is limited by the availability of appropriately supportive programming environments (Uebbing, 1987). Many times these interactions cannot be anticipated at all. Presenting rich information displays and direct access to running code often entails cluttered displays and inefficient performance. Many of these critical details and interactions cannot be analyzed before a prototype system is built. Indeed, one of the most important determinants of the success of software technologies is their amenability to revision and reimplementation on hardware and software platforms not even available when they were first developed (Brooks, 1987).

The cognitive description paradigm in HCI was a genuine advance. It provided independent conceptual foundations for the psychology of HCI that made it possible to develop useful theory. Reciprocally, it brought the HCI

domain within the purview of academic psychologists. This has opened a two-way dialog within which basic cognitive psychology may stand to gain as much from the cognitive engineering case study of HCI as HCI may stand to gain from the science of cognition (Carroll, 1987b; Norman, 1987).

4. Usability-Innervated Invention

The human factors evaluation and cognitive description paradigms share basic assumptions about the position of psychological analysis in HCI. They assume that psychology operates *outside* the development process, outside even the research prototyping process. They assume that the role of psychologists in HCI is to offer *commentary*: evaluations, theoretical descriptions, but not direct participation in the invention, design and development of new HCI technologies and artifacts. This assumed positioning and role for psychology in HCI is all the more striking when one recognizes that HCI is fundamentally a design domain. HCI is *about* designing new software tools and user interfaces. Seen in this light, the traditional paradigms for psychology in HCI have pursued a tangential, supporting role in the field's key endeavor and *raison d'etre.*

It has, of course, been recognized that serious usability research needs to pay serious attention to the nature of HCI domains and tasks. This concern has always been in the focus of HCI work. But being relevant to designer needs is not the same as taking the initiative in the design work itself. The implicit division of labor in HCI has had chronic organizational consequences. For example, a recent panel discussion at the ACM CHI'88 Conference asked how human factors specialists, and cognitive scientists working on usability, can organize to work effectively with designers and developers (Grudin, 1988). The answers offered are revealing: human factors professionals should be placed directly into development groups, human factors professionals should *manage* the developers, and usability consultants from outside the organization should be used(!). The traditional paradigms created an organizationally adversarial basis for the exchange of commentary between software developers and psychologists.

The traditionally assumed positioning and role of psychology within HCI is now being seriously questioned. In this new paradigm of "usability-innervated invention," usability is seen as connecting the invention of HCI artifacts to user needs no less essentially than nerves connect organs and muscle tissues to sensory and motor brain centers. The activity of muscles and organs is meaningful only insofar as it is innervated by sensation and action; the activity of inventing HCI artifacts is meaningful only insofar as it is innervated by usability considerations. Conversely, sensory and motor

centers exist primarily to innervate the body's muscle and organs; understanding usability is important because it produces the critical direction for HCI invention. In this view, HCI artifacts are not merely evaluated or described in terms of their usability; *they are conceived and created for usability.*

4.1 Psychology as a Mother of Invention

Building and inventing things it not a traditional activity in psychological research. Psychology is part natural science and part social science; its traditional focus is the analysis of natural and social phenomena. In the technological arena of HCI, this traditional focus was straightforwardly extended to the analysis of technology through evaluation and theoretical description. But these traditional activities also provided the opportunity for psychologists working in HCI domains to develop technological skills and domain experience. In many cases, these psychologists are now in a position not only to *analyze* usability problems, but to *synthesize* technological solutions. In his plenary address at the CHI+GI'87 Conference, Tom Landauer (1987b) succinctly captured this in casting "psychology as a mother of invention" in HCI.

Many recent prototype systems and interface techniques were invented by psychologists to instantiate specific psychological claims and to allow these claims to be explored and developed empirically. For example, Landauer's group analyzed human performance in a variety of naming and reference tasks to develop specific tools and techniques for keyword information systems (e.g., Furnas *et al.*, 1983). The database system Rabbit (Williams, 1984) and its "retrieval by elaboration" paradigm embodied claims about the structure of human memory and memory search as consisting of the manipulation of concrete exemplars. The variety of "Minimalist" training materials and software environments described in Carroll (1989) embody a set of claims about how new users learn computer applications. The display management system Rooms (Card and Henderson, 1987) embodies an analysis of typical user working sets (services and data accessed simultaneously).

User-interface metaphors are a systematic and detailed intrusion of psychology into modern computing system development (Carroll and Thomas, 1982; Carroll *et al.*, 1988). For example, systems that provide electronic workspaces that can be written to and viewed by multiple users in a cooperative interaction session are presented as "chalkboard" systems in the way that they are described to users and even in the way that they appear and operate (Stefik *et al.*, 1987). Thinking of the system as a physical chalkboard provides an initial familiarity for the user. It also suggests specific tasks and approaches to accomplishing them. It provides the user with an initial conceptual vocabulary within which to couch questions and draw conclusions. (Analogous points

could be made for other new computer interface designs ranging from task-oriented window layout (Carroll *et al.*, 1987), to object-oriented programming (Rosson and Alpert, 1988).)

Many recent structure-directed editors and intelligent tutoring systems for programming are clearly vehicles for instantiating psychological analyses of programming tasks and learning. For example, analyses of programming plans (e.g., Soloway and Ehrlich, 1984) are embodied in the Bridge tutor (Bonar and Liffick, 1987). Analyses of how students learn to program in Lisp (Anderson *et al.*, 1984) have been embodied in a variety of intelligent tutoring systems for teaching Lisp (Anderson and Skwarecki, 1986; Reiser *et al.*, 1988). Indeed, Anderson (1987) has argued that designing and evaluating computer tutors provides unique advantages to *basic*, academic psychological research into the mental procedures and knowledge that comprise human cognition.

Of course, psychologists *per se* are not always the inventors, but psychological rationale routinely plays a determining role in the invention of new software technology. In this work, HCI transcends merely serving as an arena for *applying* empirical experience and theoretical analysis to invention. A better description is that a two-way relationship has developed in which HCI artifacts themselves are treated as media for codifying experience and analysis, in which HCI theories are "applied invention" no less than HCI artifacts are "applied theory" (Carroll and Campbell, 1988). For example, the theoretical development of the concept "direct manipulation" (Shneiderman, 1983) devolved from a collection of specific HCI inventions. But this constitutes a radical shift in the underlying ontology of HCI, namely, seeing computer artifacts such as interface metaphors, menu hierarchies, programming paradigms and languages, tutors, and the like as playing theory-like roles.

One standard role of theories is to codify empirically falsifiable claims (Popper, 1965). Artifacts embody testable claims about how users can understand and make use of system function in a medium that makes appropriate empirical investigations possible. Each command name, each icon, each menu makes claims about the ways users think about the tasks they will undertake with these systems.

These claims are mutually interrelated, creating a sort of web of theory more intricate and more comprehensive than any analysis deducible from conventional discursive psychological theory. A piece of software, such as the Unix operating system, makes a huge number of specific claims about what command names, operations, and so forth will be convenient for users. These claims can be wrong (see Norman, 1981). Desktop interfaces make myriad claims about familiar presentation and natural conceptual vocabularies, about clipboards, stationery pads, folders, waste baskets—about how these objects behave and interact. Moreover, the leading claims, for example as integrated within a metaphor such as the desktop, have myriad specific

dependencies on a diverse set of ancillary claims (for example, claims inherent in the presentation of highlighting, preferences, and scrolling elevators).

Empirical theories provide explanations by placing logical and causal constraints on phenomena. Artifacts support explanations of the form "this specific feature has this specific usability consequence." The "Tear Off" command in the early Lisa desktop system provides an example. In this system, "Tear Off" spawns a new instance from a prototype object: Tear Off stationery applied to a stationery pad creates a piece of stationery. The command was a menu selection, not a gesture (Move is an example of a gestural command: one selects with the pointer and then moves by moving the pointer). Thus, there was a sort of inconsistency between Move and Tear Off. Some users initially tried to Tear Off by selecting and then rapidly sweeping the pointer (making a tearing gesture). This error has little consequence, and proved relatively easy for users to sort out on their own. A more difficult problem stemmed from the fact that Tear Off also applied to non-pad objects such as folders: the user needed to Tear Off from a "folder pad" to get a new folder (Carroll and Mazur, 1986).

Theories also contribute to the development of science by providing useful foundations for further theorizing. Artifacts facilitate theoretical development in the sense that given artifacts make task analyses possible that in turn facilitate the invention and development of new artifacts. The typewriter metaphor was a critical step in the development of the desktop metaphor, which in turn has been critical in the development of newer interface metaphors such as rooms and task maps. Understanding user problems at this level of qualitative detail can be of immediate use in the design of new software artifacts. Indeed, in subsequent desktop interface products the Tear Off command evolved into a Make New Folder command.

Theories enable and compel greater explicitness in empirical claims. This is part of the traditional motivation to formalize. Artifacts serve this role in a manner quite analogous to classical views of simulation (Fodor, 1968; Newell and Simon, 1972). To paraphrase Newell and Simon, both must "perform" the claims they incorporate: the implementation details must be made explicit, which can lead to further learning about the nature of the claims being made. Simulations, however, are used by psychologists for specific research purposes; artifacts are used by a wide range of people to do real work. Simulations are interpreted and evaluated by criteria of *descriptive adequacy* (Chomsky, 1965): a simulation of problem-solving behavior may be judged on the basis of how closely it fits the sequence of moves in a verbal protocol, whether it predicts all and only the kinds of errors that are observed, etc. Artifacts are interpreted and evaluated by criteria of *usability*.

Simulations are usually seen as convenient vehicles for theories, but not as *necessary*. Are artifacts merely convenient expressions of HCI theories, or do

they play a more fundamental role? This question cannot be answered now, but it seems likely that artifacts are in principle irreducible to a more conventional theory medium. The reason for this, if it is so, would be the unbounded interrelation of the many claims inherent in a computer artifact, the fact that everything in software seems to affect everything else (Brooks, 1987), the fact that details of context and situation critically impinge upon the usability of systems (Whiteside and Wixon, 1987; Winograd and Flores, 1986; Suchman, 1987). All these may be views of the same underlying state of affairs: the design of software may be of an order of complexity beyond that which conventional theories can explain or predict (Hayek, 1967).

In the introduction, we considered the apparent paradox that product innovations in user-interface design often *lead* HCI research rather than following from it in the conventionally assumed flow of "technology transfer" from Research to Development. However, the view of HCI in which its artifacts play theory-like roles in organizing research defuses the perplexity of this state of affairs. Empirical research often follows the explicit codification of theories. In HCI the medium of choice for expressing theories of usability is in many cases an exemplary artifact. The appearance of such an artifact predictably stimulates empirical research.

4.2 Ecological Analysis

The paradigm of usability-innervated invention has many consequences for the traditional empirical roles of psychologists working in HCI domains. There are consequences both for what kinds of situations are studied and for what kinds of information are sought in empirical studies. In both areas, the driving considerations devolve from invention. The model of research practice in experimental psychology, originally adapted to HCI through human factors evaluation, has been augmented by the requirement that empirical work bear more directly on the invention and development of new artifacts. In this sense, current work is shifting toward greater responsiveness to the ecology of HCI as an ecology of invention, design and development.

Ecologically responsive empirical analysis of HCI domains takes place *in vivo*: in software shops, more often than in psychological laboratories. It addresses *whole* problems, *whole* situations, when they are still technologically current, when their resolution can still constructively affect the direction of technological evolution. Its principal goal is the discovery of design requirements, not the verification of hypothesized direct empirical contrasts or cognitive descriptions. A recent example is the study by Curtis, *et al.* (1988) of the software design process. The detailed interviewing of real designers produced specific technical proposals for improving software tools and the coordination of project management, an assessment of major bottlenecks, and

a new framework for thinking about software design as a learning and communication process. (See Nielsen *et al.* (1986) and Rosson *et al.* (1988) for similar kinds of studies.)

Carroll and Campbell (1988) characterized HCI invention in terms of the "task-artifact cycle": a given understanding of the tasks programmers need to and want to accomplish helps to define objectives for new software artifacts (languages, environments and education, etc.) to support them in these tasks. Any artifact fundamentally alters the tasks for which it was designed, raising the need for further task analysis, and in time for the design of further artifacts, and so on. An example is the progression from user interfaces based on the typewriter metaphor to those based on the desktop. Early word-processing applications were designed to exploit specific knowledge their users already had about typewriting, function keys, data display, command names and so forth (Carroll and Thomas, 1982).

The typewriter metaphor, however, altered office tasks and in doing so helped to open up technological possibilities by preparing users for further electronic office applications (calculators, calendars, mail, database). This evolution in office task expectations and understandings was better addressed by systems employing the desktop metaphor. However, desktop systems also presented a variety of specific problems and possibilities to users (Carroll and Mazur, 1986; Whiteside *et al.*, 1985). This further task analysis has again helped to define further interface artifacts, new metaphors for display organization in user interfaces ("rooms," Card and Henderson, 1987; "task paths," Carroll *et al.*, 1987).

To operate constructively within the task-artifact cycle, HCI empirical work must provide rich analyses of real users working on real tasks. The main research setting for such ecological analysis is the case study. A case study can begin and end anywhere in the task-artifact cycle; the key requirement is access to real situations. Case-study task analysis usually consists of the collection of detailed, qualitative information (thinking-aloud protocols, interviews). Such data are arbitrarily rich: they can be returned to over and over again, and analyzed from many different perspectives. A typical approach is to make videotapes to create a vivid and permanent data library. The development of Minimalist training materials and software environments, cited earlier, was based on such case-study analysis (Carroll, 1989). Mack's (1988) inventory of new-user expectations about cause-and-effect relationships in the operation of a word processor was a case-study analysis culminating in the development of a prototype that more intuitively presented word-processing function.

It is important to collect information over a significant span of time to eliminate ephemeral effects. Monitoring patterns of actual use of a software environment often supplements the more direct interview and protocol techniques. Wixon *et al.* (1983) analyzed patterns of spontaneous interaction

with an electronic mail application to determine how to design a more usable command interface for the application. Kelley (1984) analyzed the desk calendars of office workers to determine requirements for an electronic calendar facility. Gould and Boies and their collaborators have designed a series of voice messaging systems using this approach (Gould and Boies, 1983; Gould *et al.*, 1987).

The key goal of ecological task analysis in the task-artifact cycle is to produce requirements for subsequent design work. This places emphasis on identifying big factors—big needs, big usability problems. Thus, one typical output of this phase is an error taxonomy, a qualitative description of what is giving the user trouble, how it is happening, what users are doing in consequence, etc. The complexity and rapid evolution of software technology requires richer and more open-ended methods than the direct-contrast testing of the human factors evaluation and cognitive description approaches. This richer style of task analysis is interpretive, inductive; it seeks to discover, not merely to confirm or disconfirm.

It often requires studying user-interface technologies and applications *before* they are even developed: after all, that's the point at which empirical guidance can be most effectively directive (Carroll and Campbell, 1986). For obvious reasons, it is difficult to do such work, but a variety of simulation techniques have been developed. For example, Gould *et al.* (1983) simulated a speech-recognition capability to explore technological tradeoffs in a technology that was not then available. Carroll and Aaronson (1988) analyzed interactions with a simulated intelligent-help facility to help direct the development of more usable artificial-intelligence applications.

To help direct the task-artifact cycle, new types of usability data and new roles for usability data are being developed. For example, since the ideas that lead HCI research typically become codified in products first, it is important to be able to interpret running systems, to extract key ideas and work with them. Norman (1981) made an influential psychological interpretation of key aspects of the Unix operating system. Carroll and Mazur (1986) analyzed new-user expectations and experiences using the on-line tutorial and direct-manipulation interface of the Lisa system. Rosson and Alpert (1988) have recently analyzed psychological implications of object-oriented design. Carroll *et al.* (1988) outlined tools for analyzing user-interface metaphors in design.

Another focus for the development of tools for empirical analysis is the process of software and system development. A comprehensive methodology of goal definition and measurement has been developed for guiding the discovery of appropriate usability requirements and evaluating progress toward meeting these requirements within the design process (Bennett, 1984; Carroll and Rosson, 1985; Whiteside *et al.*, 1988).

Usability-innervated invention offers a more directive role in framing new applications and user interfaces, and a more ecologically responsive role for empirical work. It incorporates and builds upon the prior orientations of human factors evaluation and cognitive description, but pushes onward in taking more seriously the fact that HCI is a design field, that it exists to invent more usable systems and software. Earlier approaches to psychology in HCI had in effect isolated the task analysis part of the task-artifact cycle from the definition, development and first use of new software and user-interface technology, because of preconceptions about the kinds of contributions psychologists might make to HCI. As a result, and in addition to a variety of specific limitations discussed above, they offered only commentary on the process and products of design, not participation.

5. The Ecology of Computing

The progression of three paradigms in the recent history of HCI comprises a case study of a field discovering what it is about. HCI has achieved much by exploiting the context of its own practice. It has assimilated the evaluation methodology of experimental psychology, the theory of cognitive science, and the invention and development of new technology. Each step in this evolution has solved some of the problems posed by the step preceding it.

The emerging paradigm of usability-innervated invention redresses the ecological limitations of direct-contrast laboratory evaluations by promoting new methods and new roles for empirical evaluation. It redresses the theoretical limitations of design by deduction by countenancing richer sources and embodiments of scientific theory. This in turn has resolved other puzzles about HCI. For example, the primacy of product-development ideas in HCI research is puzzling only until it is recognized that product development is a major context for HCI research: one of the important roles of psychology in HCI is to provide interpretation and conceptual clarification for product innovations.

Even the mysterious race between function and usability dissolves: appropriately contextualized HCI research cannot lag the technological leading edge; it lives at the technological leading edge; indeed, it creates the technological leading edge. For example, there is no race between usability and function in the development of the Rooms display management system (Card and Henderson, 1987), even though the Rooms approach is at the edge of our current understanding of display management tasks and artifacts. The race between function and usability is simply an untoward side-effect of the organizational consequences of human factors evaluation and cognitive description.

Usability-innervated invention offers a new basis for these organizational dynamics. When the basis for collaboration is evaluative or descriptive commentary offered from outside the design team, the grounds are frequently political, and power-based, or *interpreted* as political and power-based. This is completely unconstructive: it pushes empirical evaluation and psychological theory further away from invention. Operating within the task-artifact cycle as task analysts, as inventors of artifacts, offers a deeper source of interdisciplinary and inter-organizational coordination: shared understanding of what the problems are, why the current design situation is what it is, what the immediate and longer-term options are, and how they trade off. It offers the alternative of committed, cooperative work.

5.1 Science and Invention

There is a conventional view of the relationship between scientific research and the invention, design and development of practical artifacts. The idea is that basic science provides an understanding of nature which can then be applied deductively in practical contexts. The relationship between science and invention in HCI, as it has emerged through the course of the last 15 years, is interesting from this standpoint in that it appears to be culminating (at least to this point in time) somewhat unconventionally.

To be sure, the conventional view was what the field started out with: the vision of the human factors evaluation and cognitive description paradigms was to develop an empirical basis, to develop a theoretical framework and finally to apply the theory deductively in design. Through hard experience, HCI discovered that things were not this neat. Invention produces theory in HCI at least as much as it applies theory, and this has fundamentally altered the nature of the empirical work. The resolution of this may lie in a countercurrent in the history of science, questioning the conventional view itself. For example, Hindle (1981) analyzed a variety of 19th-century inventions and failed to find any deductive grounding in the basic science of the time. Hindle suggests that the conventional view may have developed as recently as the 1850s in the American scientific establishment as a tactic for increasing the prestige of and federal support for basic research.

Many well known instances of invention clearly do not conform to the conventional view. The pulley, for example, had been used effectively for some 2000 years before an adequate scientific analysis of its operation was developed within Newtonian mechanics. The violins of the 17th century were so finely crafted that their design was merely emulated for over 200 years. Indeed, only in the last couple of decades has there been any appreciable acoustic understanding of how violins really work (Hutchins, 1962). And it is not clear yet whether the science of acoustics itself was more a contributor to or a beneficiary of this work.

Of course, there *is* a relation between basic science and invention, but not a simple deductive relation. Gomory (1983) puts the point well when he argues that the development of technology is both more complex and less predictable than the basic research from which it is seen to spring. Gomory discusses the first 150 years of technology development for the steam engine. He shows that the "revolutionary" engines of the mid nineteenth century actually evolved through many small steps, each relying on the chance availability of a technological niche, an application in which the technology could survive and develop. The case study of HCI suggests that the relation between basic science and invention can be highly interactive and reciprocal. The conventional view goes wrong in trying to frame this relation too narrowly.

It is a commonplace of the philosophy of science since positivism to observe that there are no "discovery procedures," no algorithms to carry us from the raw material of empirical science to a theoretical explanation of that raw material. A way to put this point is to say analogously that there are no "invention procedures": the logical leap from basic data and theory to the invention and development of a usable artifact is neither more or less deterministic than the step we are more familiar with, namely the step from the raw material of experience to a theory of a conventional sort. The applied science of the conventional view is a myth.

Psychology is a young science, so is Computer Science, so is Cognitive Science, and above all, so is HCI. But this raises the question of whether the complex and reciprocal interaction of science and invention in HCI is attributable just to the youth of the relevant fields, to scientific growing pains as it were. In view of this possibility it is relevant to consider the acoustic analysis of the violin as conducted over the past 40 years by members of the Catgut Society, an interdisciplinary group of musicians, instrument craftsmen, physicists and engineers. Carla Maley Hutchins, the senior member of this team, told me an interesting anecdote about an early stage in her collaboration with Bell Labs physicists. The physicists' initial approach was to disassemble a violin, induce sine waves and measure resulting resonances.

It's a beautiful image; it recalls the direct contrasts of human factors evaluation and the shallow theories of cognitive description. It recalls models of error-free user behavior as bases for understanding how to design usable computer systems and applications. It is the conventional strategy of divide and conquer, which too often requires subtracting out the essence of the problem being solved. Inducing pure sine waves into the pieces of the violin to measure the resonances is not an adequate approach to understanding the violin. The sound to which a real violin responds is not a pure sine wave and it is not induced; it is a complex tone produced by bowing. Moreover, the resonances in a whole violin derive both from the parts and from the composition of the parts, indeed from the big chunk of air trapped within

the composition of the parts. Analyzing the parts, does not add up to an understanding of the behavior of the whole.

The point is not that these idealized acoustic analyses were pointless. Such work is on-going, and has even produced techniques useful in violin-making (Hutchins, 1981). And the point is not that acoustic science has nothing to offer as a foundation for understanding violins (bowing does not produce pure sine waves, but it does produce sound after all). The point is that even in physics the initial approach to applying science to design is often simplified and inadequate, whereas the effective role is more interactive and reciprocal. Indeed, the comparison can be pushed much further: the research of the Catgut Society led to the design and development of a new set of stringed instruments, the Violin Octet. The analysis could go only so far when its purview was an account of the standard string quartet (which acoustically is a very accidental collection of instruments). To develop and assess laws of acoustic scaling, to test and develop claims about the violin, it was necessary to build novel instruments (Hutchins, 1967; Hutchins and Schelleng, 1967).

The violin is intrinsically a very appealing example. But one needn't go so far. Anyone in the New York area recalls the renovation of Carnegie Hall. There was much concern and much debate about the impact this would have on the famous acoustics of that hall. Acoustics, the old science of physics, could not deductively direct or predict the outcome. Indeed, to this day the only fact that everyone agrees on is that the acoustics of Carnegie Hall are now different.

5.2 The Current Perplexity

Failure to appreciate the subtleties of technology development, coupled with the inherent limitations of the human factors evaluation and cognitive description paradigms of HCI and the emergence of the usability-innervated invention paradigm, has caused substantial perplexity in the field. One body of work has responded to Newell and Card's (1985) worry that psychology must be scientifically hard to survive in HCI by retreating into the study of low-level phenomena and of highly constrained situations creating a very insular research microcosm. One of the key areas of its focus is replicating classic phenomena from the psychology of nonsense-list learning (e.g., Polson *et al.*, 1987). This approach flaunts all the limitations of the cognitive description paradigm. It is not at all clear that it can be relevant to HCI design work.

Another body of work has rejected psychology as a totally inappropriate foundation for design work in HCI (Whiteside and Wixon, 1987; Winograd and Flores, 1986). In this view, focussing on models of the mind and conceiving of people as computational devices that process inputs, generate goal lists, and then execute plans and responses all merely obscure and

obstruct the designer's most important responsibility and objective: to understand the user's needs and wishes and to serve the user. This work flaunts the theoretical limitations of human factors evaluation, looking to hermeneutics as a conceptual foundation for design and emphasizing interpretations that are unique to the situation and to the individual doing the interpreting, and explicitly discouraging model-building or any form of abstraction. However, since it is bound to particular cases, this work cannot provide any framework for understanding HCI phenomena as types.

Both approaches are dismal in prospect: one offering no hope of practical impact and the other no hope of understanding. However, from the standpoint of the present discussion these extreme positions have despaired too quickly. An orderly evolution of HCI work has produced a paradigm that builds upon the genuine contributions of human factors evaluation and cognitive description and at the same time redresses their limitations with respect to design impact and the ecological validity of empirical work.

HCI has often been described as an "interdisciplinary" research area, but only now are the full interdisciplinary possibilities emerging. Participating fully and in a variety of roles in the evolution of computer technology offers psychologists in HCI a uniquely creative opportunity. It's a demanding opportunity. Inventing the future is more difficult than commenting on it. Pushing psychological theory to interpret and analyze new technological situations and embodying psychological claims and results in HCI artifacts is not easier than evaluating finished systems, computing *t*-tests and calculating performance times. But then one does not move to the frontier for the comforts of familiarity. The possibility and the challenge of HCI today is to move forward to new roles and new ideas in technology and science.

ACKNOWLEDGEMENT

This paper is derived from lectures given at Teachers College, Columbia University, the University of Michigan, the University of Western Ontario, and the IBM Watson Research Center in the winter of 1988, and from collaborative discussions with Robert Campbell and Elliot Soloway. I am grateful to Norman Brown, John Karat, Wendy Kellogg, Joan Roemer, Mary Beth Rosson and Linda Tetzlaff for comments on an earlier version.

REFERENCES

Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. *Memory and Cognition* **9**, 422–433.

Anderson, J. R. (1987). Methodologies for studying human knowledge. *Brain and Behavioral Sciences* **10** (3), 467–505. (With commentary)

Anderson, J. R., and Skwarecki, E. (1986). The automated tutoring of introductory computer programming. *Comm. ACM* **29**, 842–849.

Anderson, J. R., Farrell, R., and Sauers, R. (1984). Learning to program in Lisp. *Cognitive Science* **8,** 87–129.

Barnard, P. J., Hammond, N. V., Morton, J., Long, J. B., and Clark, I. A. (1981). Consistency and compatibility in human-computer dialog. *Int. J. Man-Machine Studies* **15,** 87–134.

Bennett, J. L. (1984). Managing to meet usability requirements: Establishing and meeting software development goals. *In* "Visual display terminals" (J. Bennett, J. Sandelin, and M. Smith, eds.), pp. 161–184. Prentice-Hall, Englewood Cliffs, New Jersey.

Black, J. B., and Sebrechts, M. M. (1981). Facilitating human-computer communication. *Applied Psycholinguistics* **2,** 149–177.

Bonar, J. G., and Liffick, B. W. (1987). A visual programming language for novices. University of Pittsburgh Technical Report LSP-5.

Brooks, F. P. (1987). No silver bullet: Essence and accidents of software engineering. *IEEE Computer* **20,** 10–19.

Card, S. K., and Henderson, D. A. (1987). A multiple virtual-workspace interface to support user task switching. *Proc. CHI + GI'87: Human Factors in Computing Systems and Graphics Interface*, pp. 53–59.

Card, S. K., English, W. K., and Burr, B. J. (1978). Evaluation of mouse, rate-controlled isometric joystick, step keys, and task keys for text selection on a CRT. *Ergonomics* **21** (8), 601–613.

Card, S. K., Moran, T. P., and Newell, A. (1983). "The Psychology of Human-Computer Interaction." Erlbaum, Hillsdale, New Jersey.

Carlisle, J. H. (1970). Comparing behavior at various computer display consoles in time-shared legal information. Rand Corporation, Report No. AD712695. Santa Monica, California.

Carroll, J. M. (1982). Learning, using and designing command paradigms. *Human Learning: Journal of Practical Research and Applications* **1,** 31–63.

Carroll, J. M. (1985). "What's in a Name? An Essay in the Psychology of Reference." W. H. Freeman, New York.

Carroll, J. M. (1987a). Five gambits for the Advisory Interface Dilemma. *In* "Psychological Issues of Human Computer Interaction in the Work Place" M. Frese, U. Ulich, and W. Dzida, pp. 257–274. North-Holland, Amsterdam.

Carroll, J. M., ed. (1987b). "Interfacing Thought: Cognitive Aspects of Human Computer Interaction." Bradford Books/M.I.T. Press, Cambridge, Massachusetts.

Carroll, J. M. (1988). Modularity and naturalness in cognitive science. *Metaphor and Symbolic Activity* **3** (2), 61–86.

Carroll, J. M. (1989). "The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill," To be published.

Carroll, J. M., and Aaronson, A. P. (1988). Learning by doing with simulated intelligent help. *Comm. ACM* **31,** 1064–1079.

Carroll, J. M., and Campbell, R. L. (1986). Softening up Hard Science: Reply to Newell and Card. *Human-Computer Interaction* **2,** 227–249.

Carroll, J. M., and Campbell, R. L. (1988). Artifacts as psychological theories: The case of human-computer interaction. IBM Research Report RC 13454. Yorktown Heights, New York. To appear in 1989 in *Behavior and Information Technology* **8**.

Carroll, J. M., and Mazur, S. A. (1986). Lisa Learning. *IEEE Computer* **19** (11), 35–49.

Carroll, J. M., and Rosson, M. B. (1985). Usability specification as a tool in interactive development. *In* "Advances in Human-Computer Interaction 1" (H. Hartson, ed.) pp. 1–28. Ablex, Norwood, New Jersey.

Carroll, J. M., and Soloway, E. (1988). The evolving role of software psychology in software development practice. Unpublished manuscript, IBM Watson Research Center, Yorktown Heights, New York.

Carroll, J. M. and Thomas, J. C. (1982). Metaphor and the cognitive representation of computing systems. *IEEE Trans. Systems, Man and Cybernetics* **12,** 107–115.

Carroll, J. M., Herder, R. E., and Sawtelle, D. S. (1987). TaskMapper. *Human-Computer Interaction: Proceedings of INTERACT'87*, pp. 973–978.

Carroll, J. M., Mack, R. L., and Kellogg, W. A. (1988). Interface metaphors and user interface design. *In* "Handbook of Human-Computer Interaction" (M. Helander, ed.). North Holland, Amsterdam, pp. 67–85.

Chapanis, A. (1959). "Research Techniques in Human Engineering." The Johns Hopkins Press, Baltimore, Maryland.

Chase, W. C., and Simon, H. A. (1973). Perception in chess. *Cognitive Psychology* **4,** 55–81.

Chi, M. T., Glaser, R., and Farr, M. J., eds (1988). "The Nature of Expertise." Erlbaum, Hillsdale, New Jersey.

Chomsky, A. N. (1965). "Aspects of the Theory of Syntax." MIT Press, Cambridge, Massachusetts.

Crowder, R. G. (1976). "Principles of Learning and Memory." Erlbaum, Hillsdale, New Jersey.

Curtis, B. (1980). Measurement and experimentation in software engineering. *Proc. IEEE* **68** (9), 1144–1157.

Curtis, B. (Ed.) (1985) "Human Factors in Software Development." IEEE Computer Society Press, Washington, D. C.

Curtis, B. (1986). By the way, did anyone study any real programmers? *In* "Empirical Studies of Programmers" (E. Soloway and S. Iyengar, eds.), pp. 256–262. Ablex, Norwood, New Jersey.

Curtis, B., Krasner, H., and Iscoe, N. (1988). A field study of the software design process for large systems. *Comm. ACM* **31,** 1268–1287.

Dijkstra, E. W. (1972). Notes on structured programming. *In* "Structured Programming" (O. J. Dahl, E. W. Dijkstra and C. A. R. Hoare, eds.), pp. 1–82. Academic Press, New York.

Dreyfus, H. L., and Dreyfus, S. E. (1986). "*Mind over Machine: The Power of Human Intuition and Expertise in the Era of the Computer.*" The Free Press, New York.

Esper, E. A. (1925). A technique for the experimental investigation of associative interference in artifical linguistic material. *Language Monographs* **1,** 1–47.

Fodor, J. A. (1968). "Psychological Explanation." Random House, New York.

Furnas, G. W., Landauer, T. K., Gomez, L. M., and Dumais, S. T. (1983). Statistical semantics: Analysis of the potential performance of key-word information systems. *Bell System Technical J.*, **62** 1753–1806.

Gleick, J. 1987. "Chaos: Making a New Science." Viking, New York.

Gomez, L. M., and Lochbaum, C. C. (1985). People can retrieve more objects with enriched keyword vocabularies. But is there a performance cost? *In* "Human-Computer Interaction—INTERACT'84" (B. Shackle, ed.), pp. 257–261. North Holland, Amsterdam.

Gomory, R. E. (1983). Technology development. *Science* **220,** 576–580.

Gould, J. D., and Boies, S. J. (1983). Human factors challenges in creating a principal support office system—The Speech Filing System approach. *ACM Trans. Office Information Systems* **1** (4), 273–298.

Gould, J. D., and Lewis, C. H. (1985). Designing for usability: Key principles and what designers think. *Comm. ACM* **28** (3), 300–311.

Gould, J. D., Boies, S. J., Levy, S., Richards, J. T., and Schoonard, J. (1987). The 1984 Olympic Message System: A case study of system design. *Comm. ACM* **30,** 758–769.

Gould, J. D., Conti, J., and Hovanyecz, T. (1983). Composing letters with a simulated listening typewritter. *Comm. ACM* **26** (4), 295–308.

Green, P. (1987). Tips on writing a good paper proposal. *Computer Systems Technical Group Bull.* **14,** 6–10.

Grudin, J. 1988. Integrating human factors in software development. *CHI'88 Conference on Human Factors in Computing Systems*, pp. 157–160.

Hauptmann, A. G., and Green, B. F. (1983). A comparison of command, menu-selection and natural language compter programs. *Behaviour and Information Technology* **2,** 163–178.

Hayek, F. A. (1967). The theory of complex phenomena. *In* "Studies in Philosophy, Politics, and Economics" (F. A. Hayek, ed.). University of Chicago Press, Chicago.

Hindle, B. (1981). "Emulation and Invention." New York University Press, New York.

Holt, R. W., Boehm-Davis, D. A., and Schultz, A. C. (1987). Mental representations of programs for student and professional programmers. *In* "Empirical Studies of Programming: Second Workshop" (G. M. Olson, S. Sheppard, and E. Soloway, eds.), pp. 33–46. Ablex, Norwood, New Jersey.

Hutchins, C. M. (1962). The physics of violins. *Scientific American*, November (Reprint 289).

Hutchins, C. M. (1967). Founding a family of fiddles. *Physics Today* **20** (2).

Hutchins, C. M. (1981). The acoustics of violin plates. *Scientific American* **245** (4), 170–186.

Hutchins, C. M. and Schelleng, J. C. (1967). A new concert violin. *J. Audio Engineering Soc.* **15** (4).

Kelley, J. F. (1984). An iterative design methodology for user-friendly natural language office information applications. *ACM Trans. Office Information Systems* **2**, 26–41.

Landauer, T. K. (1987a). Relations between cognitive psychology and computer system design. *In* "Interfacing Thought: Cognitive Aspects of Human-Computer Interaction" (J. M. Carroll, ed.), pp. 1–25. Bradford/MIT Press, Cambridge, Massachusetts.

Landauer, T. K. (1987b). Psychology as a mother of invention. *Proc. CHI + GI'87: Human Factors in Computing Systems and Graphics Interface*, pp. 333–335.

Landauer, T. K., Galotti, K. M., and Hartwell, S. (1983). Natural command names and initial learning: A study of text editing terms. *Comm. ACM* **26**, 495–503.

Ledgard, H., Whiteside, J. A., Singers, A., and Seymour, W. (1980). The natural language of interactive systems. *Comm. ACM* **23**, 556–563.

Liebelt, L. S., McDonald, J. E., Stone, J. D., and Karat, J. (1982). The effect of organization on learning menu access. *Proc. Human Factors Society, 26th Annual Meeting*, pp. 546–550.

Love, T. (1977). Relating Individual Differences in Computer Programming Performance to Human Information Processing Abilities. Ph. D. Dissertation, University of Washington.

Mack, R. L. (1988). Understanding and learning text-editing skills: Observations on the role of new user expectations. *In* "Cognition, Computing and Cooperation" (S. Robertson, J. Black, and W. Zachary, eds.). Ablex Publishing, Norwood, New Jersey.

MacGregor, J. N. and Lee, E. S. (1987). Performance and preference in videotex menu retrieval: A review of the empirical literature. *Behavior and Information Technology* **6**, 43–68.

McKeithen, K. B., Reitman, J. S., Reuter, H. H., and Hirtle, S. C. (1981). Knowledge organization and skill differences in computer programmers. *Cognitive Psychology* **13**, 307–325.

Miller, G. A. (1956). The magical number seven plus or minus two: Some limits on our capacity for processing information. *Psych. Rev.* **63**, 81–97.

Moran, T. P. (1981). The command language grammar: A representation for the user interface of interactive computer systems. *Int. J. Man-Machine Studies* **15**, 3–50.

Murrel, S. (1983). Computer communication system design affects group decision making. *Proc. CHI'83 Human Factors in Computing Systems*, pp. 63–67.

Newell, A. (1973). You can't play twenty questions with nature and win. In "Visual Information Processing" (W. Chase, ed.). Academic Press, New York.

Newell, A., and Card, S. K. (1985). The prospects for psychological science in human-computer interaction. *Human-Computer Interaction* **1**, 209–242.

Newell, A., and Simon, H. A. (1972). "Human Information Processing." Prentice-Hall, Englewood Cliffs, New Jersey.

Nielsen, J., Mack, R. L., Bergendorff, K., and Grischkowsky, N. L. (1986). Integrated software usage in the professional work environment: evidence from questionnaires and interviews. *Proc. CHI'86 Human Factors in Computing Systems*, pp. 162–167.

Norman, D. A. (1981). The trouble with Unix. *Datamation* **27**, 556–563.

Norman, D. A. (1987). Cognitive Engineering—Cognitive Science. *In* "Interfacing Thought:

Cognitive Aspects of Human-Computer Interaction" (J. M. Carroll, ed.), pp. 323–336. Bradford/MIT Press, Cambridge, Massachusetts.

Paivio, A. (1971). "Imagery and Verbal Processes." Holt, Rinehart & Winston, New York.

Polson, P. (1987). A quantitative theory of human-computer interaction. *In* "Interfacing Thought: Cognitive Aspects of Human-Computer Interaction." (J. M. Carroll, ed.), pp. 184–235. Bradford/MIT Press, Cambridge, Massachusetts.

Polson, P., Kieras, D., and Muncher, E. (1987). Transfer to skills between inconsistent editors. Microelectronics and Computer Technology Corporation Technical Report ACA-HI-395-87, Austin, Texas.

Popper, K. (1965). "Conjectures and Refutations." Harper and Row, New York.

Postman, L., and Stark, K. (1962). Retroactive inhibition as a function of set during the interpolated task, *J. Verbal Learning and Verbal Behavior* **10,** 44–51.

Pylyshyn, Z. (1973). What the mind's eye tells the mind's brain: A critique of mental imagery. *Psych. Bull.* **80,** 1–24.

Reiser, B. J., Friedman, P., Gevins, J., Kimberg, D. Y., Ranney, M., and Romero, A. (1988). A graphical programming language interface for an intelligent Lisp tutor. Princeton University CSL Report 15.

Robertson, G., McCracken, D., and Newell, A. (1981). The ZOG approach to man-machine communication. *Int. J. Man-Machine Communication* **14,** 461–488.

Rosson, M. B., and Alpert, S. (1988). The cognitive consequences of object-oriented design. IBM Research Report RC 14191. Yorktown Heights, New York.

Rosson, M. B., Maass, S., and Kellogg, W. A. (1988). The designer as user: Building requirements for design tools from design practice. *Comm. ACM* **31,** 1288–1298.

Sheil, B. A. (1981). The psychological study of programming. *ACM Computing Surveys* **13,** 101–120.

Sheppard, S. B., Curtis, B., Millman, P., and Love, T. (1979). Modern coding practices and programmer performance. *IEEE Computer* **12** (12), 41–49.

Shneiderman, B. (1980). "Software Psychology: Human Factors in Computer and Information Systems." Winthrop, Cambridge, Massachusetts.

Shneiderman, B. (1983). Direct manipulation: A step beyond programming languages. *IEEE Computer* **16** (8), 57–69.

Shneiderman, B., and McKay, D. (1976). Experimental evaluations of computer program debugging and modification. *Proc. 6th Int. Congress of the Int. Ergonomics Assoc.*

Soloway, E., and Ehrlich, K. (1984). Empirical studies of programming knowledge. *IEEE Trans. Software Engineering* **SE-10** (5), 595–609.

Stefik, M., Foster, G., Bobrow, D., Kahn, K., Lanning, S., and Suchman, L. (1987). Beyond the chalkboard: Computer support for collaboration and problem solving in meetings. *Commu. ACM* **30,** 32–47.

Suchman, L. (1987). "Plans and Situated Actions." Cambridge University Press, Cambridge.

Tennant, H. R., Ross, K. M., and Thompson, C. W. (1983). Usable natural language interfaces through menu-based natural language understanding. *Proc. CHI'83 Human Factors in Computing Systems*, pp. 154–160.

Uebbing, J. Panel on making products. *Proc. OOPSLA'87: Object-Oriented Programming Systems, Languages and Applications.* Special issue of *Sigplan Notices* **23** (5), 1988, pp. 105–111.

Walther, G. H., and O'Neil, H. F. (1974). On-line user-computer interface: the effects of interface flexibility, terminal type, and experience on performance. *Proc. National Computer Conf.* **43.**

Weissman, L. (1974). A Methodology for Studying the Psychological Complexity of Computer Programs. Ph.D. Dissertation, University of Toronto.

Whiteside, J., and Wixon, D. (1987). Improving human-computer interaction—a quest for cognitive science. In "Interfacing Thought: Cognitive Aspects of Human-Computer Interaction" (J. M. Carroll, ed.), pp. 337–352. Bradford/MIT Press, Cambridge, Messachusetts.

Whiteside, J., Bennett, J., and Holtzblatt, K. (1988). Usability engineering: Our experience and evolution. *In* "Handbook of Human-Computer Interaction" (M. Helander, ed.). North Holland, Amsterdam.

Whiteside, J., Jones, S., Levy, P. S., and Wixon, D. (1985). User performance with command, menu, and iconic interfaces. *Proc. CHI'85: Human Factors in Computing Systems*, pp. 185–191.

Williams, M. D. (1984). What makes RABBIT run? *Int. J. Man-Machine Studies* **16**, 405–438.

Winograd, T., and Flores, F. (1986). "Understanding Computers and Cognition: A New Foundation for Design." Ablex, Norwood, New Jersey.

Wixon, D., Whiteside, J., Good, M., and Jones, S. (1983). Building a user-defined interface. *Proc. CHI'83 Human Factors in Computing Systems*, pp. 24–27.

This Page Intentionally Left Blank

Protocol Engineering

MING T. LIU*

Department of Computer and Information Science
The Ohio State University
Columbus, Ohio

* This work was supported by U.S. Army CECOM, Ft. Monmouth, NJ, under Contract No. DAAB07-88-K-A003. The reviews, opinions, and/or findings contained in this article are those of the author and should not be construed as an official Department of the Army position, policy or decision.

Copyright © 1989 by Academic Press, Inc.
All rights of reproduction in any form reserved.
ISBN 0-12-012129-8

1. Introduction

Recent advances in microelectronics and rapid developments in information technology have made computer networking and distributed processing possible. As a result, many computer-communication networks have been designed, implemented, and put into service around the world during the past decade (Stalling, 1988; Tanenbaum, 1988). They range from a few connected personal computers to a complex interconnection of thousands of computers, using a wide variety of communication media such as twisted wire pairs, coaxial cables, optical cables, microwave links and satellite channels. Worldwide electronic mail is now a daily reality for millions of people, and networks have become an essential tool for many users in academia, business, industry and government.

Depending on specific applications and circumstances, the communication between a pair of end users in a computer network may take several different forms. For example, a terminal user may invoke a remote applications program, and two application programs in different hosts may interact with each other. To enable an orderly exchange of information between physically separated computers, a set of rules is required to govern the interaction between the communicating entities. These rules are collectively called computer-communication protocols, or *protocols* for short.

Protocols are simply a set of rules prescribing the manner in which communication takes place, the meaning of information exchanged and the appropriateness of communication under prescribed conditions. At the lowest level, protocols may prescribe how information is to be transmitted and received over a physical medium, and how that information is to be physically represented on the medium. At higher levels, protocols aim to overcome inherent unreliability in low levels, to prevent congestion and deadlocks, to control the flow of information, and to provide mechanisms for delivery, addressing and routing of messages. At still higher levels, protocols may provide services for transferring files between physically separated computers, for enabling communication between incompatible terminals, for ensuring

security in data transmission, etc. Therefore, protocols play an important role in computer networks, and form the cornerstone upon which computer networks are built.

Because protocols are the rules defining the interaction between communicating entities residing at different nodes of the network, running in parallel, and communicating through possibly unreliable channels, their design is always a challenging problem. In the last two decades, informal techniques used to design these protocols have been largely successful, but have also yielded a disturbing number of errors or unexpected and undesirable behavior in those protocols (Bochmann and Sunshine, 1980). Consequently, formal methods of protocol design have emerged in the last decade. Using the formal approach, a protocol is represented by a formal model (or interchangeably, a formal specification). Analytic techniques are then used to examine logical correctness and performance of the protocol before it is actually implemented. The methodology has been proved to be so effective in identifying many protocol design errors that the discipline in this area is now called *protocol engineering* and is currently receiving more and more attention from both industry and academia.

Referring to Fig. 1, one can see the domain of protocol engineering as a system that allows a protocol designer to specify a protocol formally, to test this specification for correctness (validation of syntax and verification of semantics), to obtain some early indication of how it would perform (efficiency), to compile major parts of the implementation directly from the formal specification, and to test the resultant implementation for conformance to its specification (Rudin, 1985).

With the proliferation of different network architectures, protocol conversion is needed to achieve interoperability between processes that implement different protocols. How should it be done? How can one prove that a conversion is correct? What is meant by a correct conversion? Again, formal methods have been recently proposed to tackle these problems (Green, 1986; Lam, 1986). Thus, protocol conversion can be included in the domain of protocol engineering.

Another area of interest in protocol engineering is protocol synthesis. Protocol analysis and protocol synthesis are two inherently different but complementary approaches to ensuring the correctness of communication protocols (Zafiropulo *et al.*, 1980). In the analysis approach, an already designed protocol is first examined to reveal some properties, desirable or undesirable, and then modified to get rid of the undesirable ones; in the synthesis approach, rules ensuring some desirable properties are enforced during the protocol design process. The synthesis approach has the advantage over the analysis approach in that it can assist the protocol designer to reduce the possibility of making errors, if not to prevent it totally, during the protocol design process.

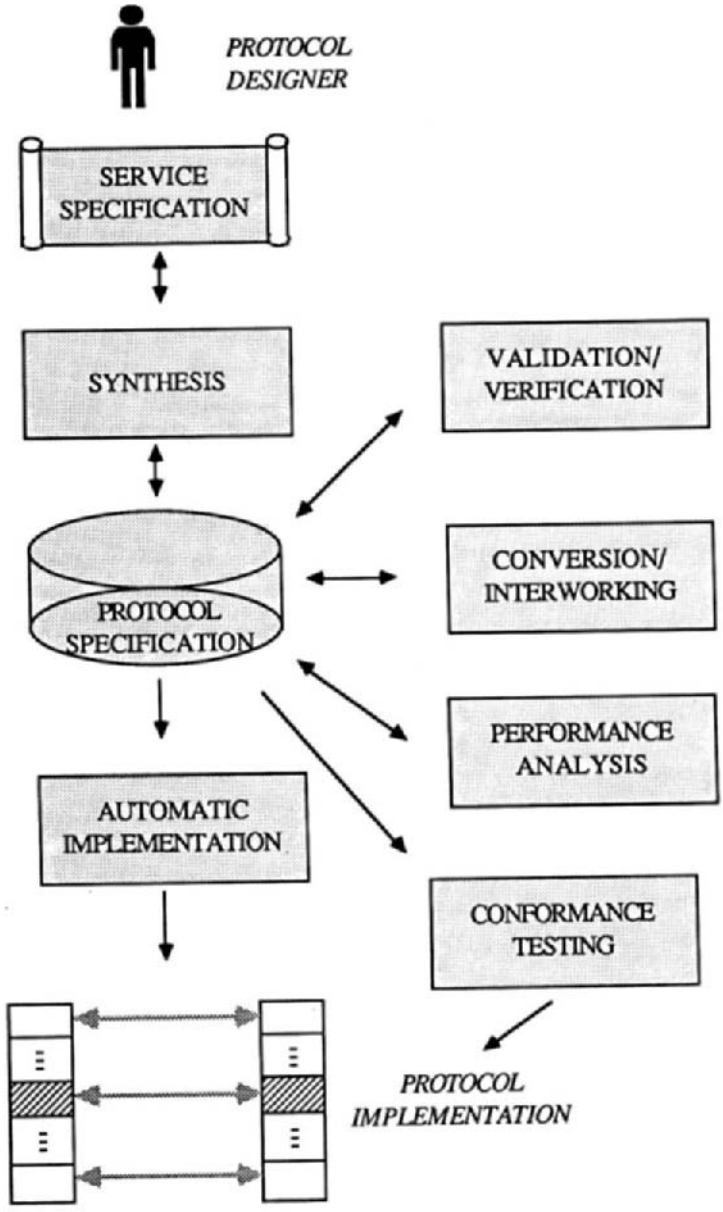


FIG. 1. Various aspects of protocol engineering

This article presents both an expository survey and our research results in protocol engineering. Because of the length of the presentation, no attempt has been made to describe all the published work in the literature or to give a detailed comparison of our results with those of other researchers. It is hoped that the presentation here and the associated discussion will enable the reader to understand the current state of the art in protocol engineering (Zimmermann, 1983).

In passing, it is worth noting that Piatkowski was probably the first person to coin the term protocol engineering at an IFIP-sponsored workshop on computer-network protocols in 1981 (Piatowski, 1981). Just like the field of software engineering that took about 20 years to mature, some progress has been made in protocol engineering since 1981, but it has been slow and tedious (Piatowski, 1983, 1986; Rudin, 1985, 1988). Nevertheless, with the advent of the Integrated Services Digital Network (ISDN), which is a projected worldwide public computer-communication network that will provide a wide variety of services (such as voice, data, video, fax, and image transmissions) to end users in the 1990s, it is clear that protocol engineering will play an active and important role in the development and implementation of the ISDN (Duc and Chew, 1986).

2. Network Architecture

Modern computer networks are designed in a highly structured way, first to reduce their design complexity, and second to increase their modifiability—the ability to change the implementation of a module without affecting the other modules as long as the interface between modules remains constant. Therefore, most networks are organized as a hierarchy of layers, each one being built on its immediate lower layer. The function of each layer is to offer certain services to the higher layer, shielding the higher layers from knowing the details of how these services are actually implemented. In this section we briefly present a set of common terminologies, concepts and conventions that will be used in this article.

2.1 OSI Reference Model

Facilitating communications between information processing systems in a heterogeneous environment requires a universal framework of computer networking architecture. It is for this purpose that the International Organization for Standardization (ISO) initiated development of worldwide standards for the creation of an *open system environment*. When complying with these standards, an information system would be open to communicate

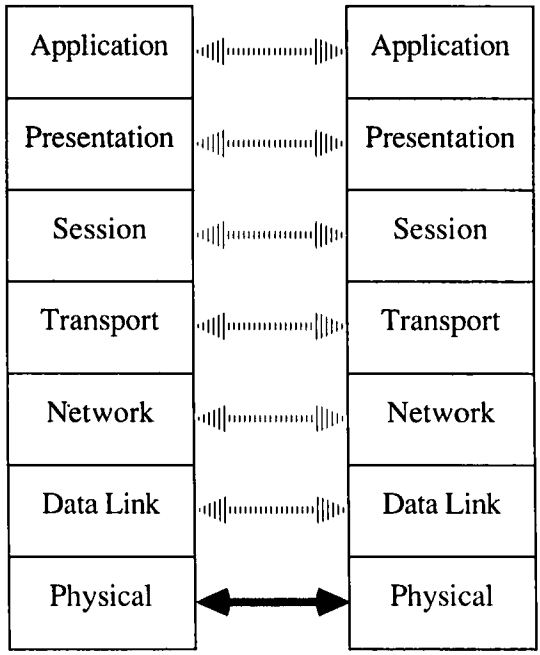


FIG. 2. A network architecture based on the OSI reference model

with any other system conforming to the same standards. After several years of efforts by ISO (Day and Zimmermann, 1983), the result of this standardization is the well-known seven-layer Open Systems Interconnection (OSI) Reference Model (See Fig. 2). It provides a common basis to guide future development of mutually compatible information processing systems that will greatly benefit both computer vendors and users.

The reference model has seven layers. In the following we will briefly discuss each layer of the architecture in turn, beginning with the bottom layer. (For a more thorough presentation of the OSI Reference Model, see Stallings (1988) and Tanenbaum (1988).) The lowest layer—the physical layer—provides the electrical, mechanical, functional, and procedural details necessary to transmit raw bits over a communication channel. The transmission form within the physical layer is transparent to the data-link layer and higher layers. The purpose of the data-link layer is to transform a raw transmission channel into a line that is free of transmission errors to the network layer. It provides mechanisms for error recovery due to transmission noise burst, damage, loss, and duplication. The network layer ensures that all data are correctly received at their destinations, and in the proper order. It also controls the routing of data in the network, and prevents congestion and deadlocks. The transport layer provides multiplexing services, and handles important issues such as

naming and addressing, connection establishment and termination, buffering and flow control, error recovery, and synchronization. The session layer establishes connections between users (sessions) and manages them. Unlike the first five lower layers, which are necessary for the correct operation of the network, the purpose of the presentation layer is to provide certain useful but not always essential services such as text compression, cryptographic transformations, data security, communication between incompatible terminals, and file transfer. Finally, the top of the hierarchy—the application layer—directly provides services to the users of the network.

In addition to ISO, a number of national standard organizations (such as NIST, formerly NBS, and ANSI) and international standard organizations (such as CCITT and ECMA) have been taking part in the development of the OSI Reference Model. Although many existing protocol architectures vary from the layered structure of the ISO Reference Model, the layered approach has become essentially universal and has been widely adopted in many computer networks such as the IBM Systems Network Architecture (SNA) and the DEC Digital Network Architecture (DNA) (see Stalling, 1988; Tanenbaum, 1988).

2.2 Layering and Abstraction

The major contributions of the OSI work are not only the creation of a common framework for intersystem communications but also the defining of a set of terminologies, conventions, and concepts so that research work in the literature can be stated in and interpreted through a common glossary. In the area of formal specification and verification, the concepts of service and protocol are crucial.

While Fig. 2 illustrates a layered protocol structure, Fig. 3 shows in more detail a particular layer (layer N) and its interaction with the layers above and below (layers $N + 1$ and $N - 1$). In a computer network, each layer consists of a collection of protocol entities (or protocol processes) that are distributed over different locations. The protocol entities that are in the same layer are called peer entities (peer processes) or communicating entities (communicating processes). The peer entities of layer N provide the communication services (called N-services) to layer $N + 1$ users. The services provide by layer N are accessed by the user entities through a layer interface. Likewise, the entities of layer N access the communication services, called $(N - 1)$ services, provided by the layer below through another layer interface. The entities of layer N use these services for exchanging messages. The rules that govern the exchange of these messages among the entities are collectively called an N-protocol.

In the context of the OSI Reference Model, it is important to distinguish between two independent notions: layers of functions and levels of abstractions.

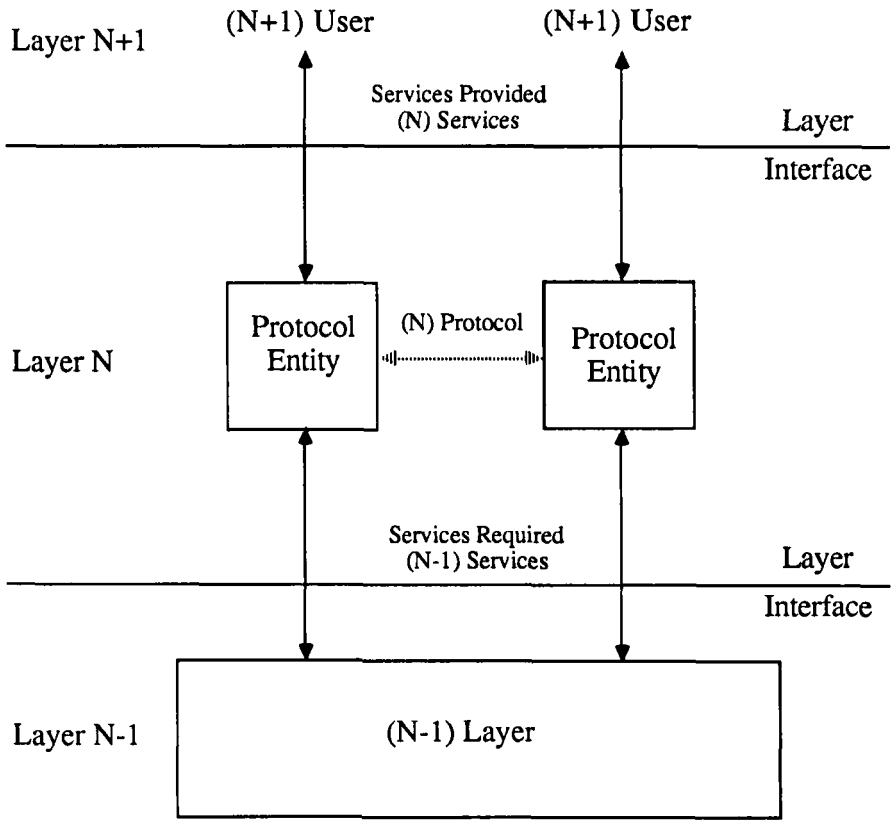


FIG. 3. Structure of a protocol layer

Layering is a structuring technique by which a system can be logically decomposed into smaller subsystems. In the OSI Reference Model, the layering approach subdivides the functionality of an open system into seven layers, each responsible for a specific set of functions. This approach has at least two significant advantages.

1. The whole system is subdivided into individual pieces of manageable size that are more comprehensible and subject to independent implementation and maintenance.
2. A portion of the system is able to perform its function before the completion of the other parts. This is especially important in establishing standards. As we can see, at the present time, while the lower layers of the

OSI model have already been developed and become functioning, the standardization of the upper layers is still in process.

Abstraction is an architectural concept applying to all layers of an open system. For each layer N, there are two levels of abstractions—(N)-Service and (N)-Protocol. At the higher level of abstraction, (N)-Service defines the interface between (N)-layer and $(N+1)$-layer. At the lower level of abstraction, (N)-Protocol defines the behavior of (N)-entities inside (N)-layer.

As illustrated in Fig. 4, from the viewpoint of $(N-1)$-layer, (N)-Service represents the capability of the (N)-layer and *all* the layers below; it is not

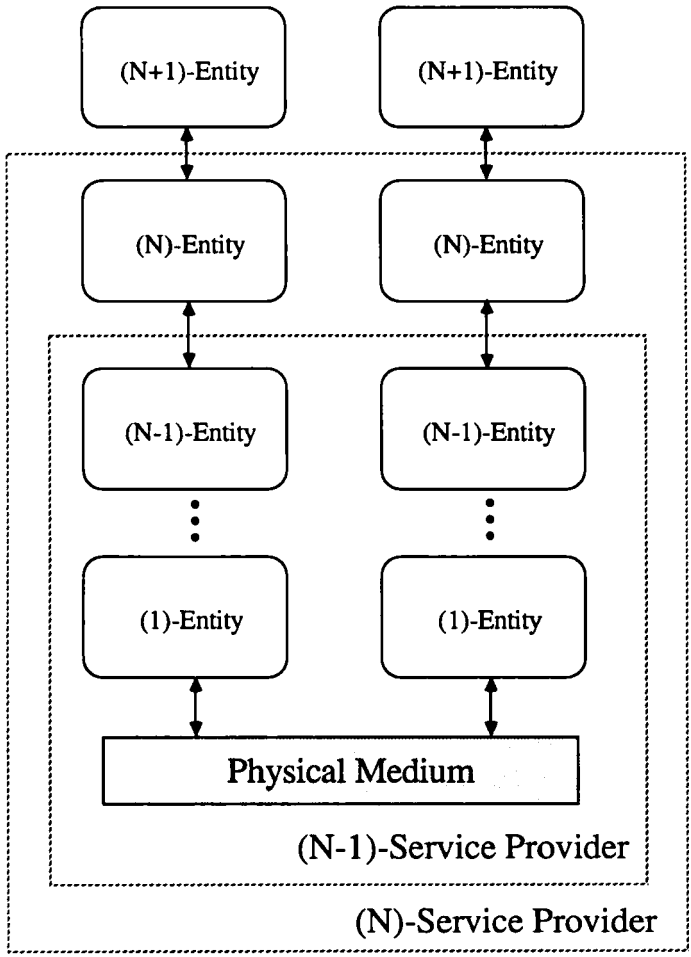


FIG. 4. OSI architecture

concerned about how the capability is realized. The (N)-entities, when making use of their underlying ($N-1$)-Service, constitute a *logical implementation* of the (N)-Service. The use of abstraction has several advantages:

1. Each layer, knowing the service provided from its lower layer, can be designed and developed with little knowledge of the internal operations in the lower layers.
2. The effect of any future changes of a protocol is localized within a layer provided that the service offered to the higher layer remains the same.

In reality, no data or messages are transmitted horizontally from one entity to another except in the lowest layer. Instead, each layer passes data down to the layer immediately below it, until the lowest layer is reached. Through the services provided by the layer immediately below it, however, each entity is able to conceptually think of its communication as being horizontal.

2.3 Protocol and Service Specifications

Specification refers to the information that is used to describe an object. It should describe only those requirements that the object must satisfy, and no more. With respect to the protocol architecture mentioned above, there are two kinds of specifications in each layer N of the protocol hierarchy:

A. The *N-service specification* describes what services the layer N protocol entities provide for their users in the $N+1$ protocol layer. The services provided by a protocol layer are usually based on a set of service primitives which describes the operations at the interface through which the services are provided.
B. The *N-protocol specification* describes the interactions among the layer N protocol entities. The interactions are defined in terms of the services provided to layer $N+1$, and the services available from layer $N-1$.

Most work on formal techniques for specifying communication protocols has concentrated on protocol specifications and not on service specifications. However, service specification is receiving more and more attention in current protocol design (See Section 5.1). Several major formal models primarily used for protocol specification will be presented in the next section.

3. Formal Models for Protocol Specification

In order to specify a protocol, one must describe what the protocol should do and how the protocol should react to external stimuli such as service primitives. The implementation of a protocol is an implicit specification, i.e.,

the protocol is specified to behave exactly as does the implementation. Since most protocols are very complex, one prefers to specify a protocol abstractly during the initial stage of design and to leave until a later stage those implementation details that do not affect the function of how the protocol should behave. The main objective of the abstraction is to facilitate the validation and verification of the protocol for its correctness before its actual implementation. As mentioned previously, conventional methods of informal narrative specification have demonstrated their shortcomings as protocol design errors crop up (Bochmann and Sunshine, 1980). In this section we present a brief survey of important formal models that have been proposed for protocol specification. For comparison we will use the alternating bit protocol (Lynch, 1968; Bartlett *et al.*, 1969) as a common example for eight of the most widely used models.

The formal methods discussed in this section fall into three main categories. The first category includes state-transition models such as finite-state automata (FSA), formal grammars, and Petri nets. The second category includes programming language models such as abstract programs, temporal logic, and abstract data types. In the third category are hybrid models that include both states and language constructs in the specification of protocols.

3.1 State-Transition Models

The state-transition model is motivated by the observation that protocols can be modeled by event-driven processes (entities) that communicate with each other through message passing. The various protocol models differ in the way processes are specified. Models falling into this category include finite-state automata, formal grammars, and Petri nets and their derivatives. The state-transition model of one sort or another with such events forming its inputs is very natural and easy to automate. However, for realistic protocols of any complexity, the number of events and states can become unworkably large, thereby creating the so-called *state explosion* problem.

3.1.1 Finite-State Automata (FSA)

FSA models are one of the earliest formal models to be applied to protocols. Ever since Lynch (1968) and subsequently Bartlett *et al.* (1969) used FSA for specifying the Alternating Bit Protocol (ABP), the number of formal models for protocol specification has increased at a rapid rate. The ABP has since then become a classical example and been used extensively by other models to illustrate their feasibility in protocol specification and verification.

FSA models are based on the observation that protocols consist largely of relatively simple processing activities in response to a number of *events* such as commands from the user, message arrivals from another peer entity, and

internal timeouts. Therefore, finite-state automata with such events forming their transitions are a natural model for specifying communication protocols. The basic approach is to specify the communication system as a collection of finite-state automata, each describing the behavior of a communicating entity.

In this model, a protocol is represented by a network of communicating finite-state automata, in which the behavior of a protocol entity is modeled by a finite-state automaton and the channels between protocol entities are modeled by FIFO queues. Each state of the finite-state automaton corresponds to a different control stage of the entity. Each transition of the automaton is labeled with either an input event that enables the transition or an output event that takes place as part of the transition.

Figure 5 illustrates an FSA model for specifying the Alternating Bit Protocol (ABP). The protocol provides reliable transmission of data from one communicating entity (called the sender) to the other (called the receiver). It uses a frame-oriented transmission technique: data are divided into frames, and frames are transmitted one at a time. Transmission errors and losses, which must be detected, are recovered by the protocol. The sender sends a data frame together with a control bit, which alternates in value between successive data frames, and waits for an acknowledgment frame from the receiver. The

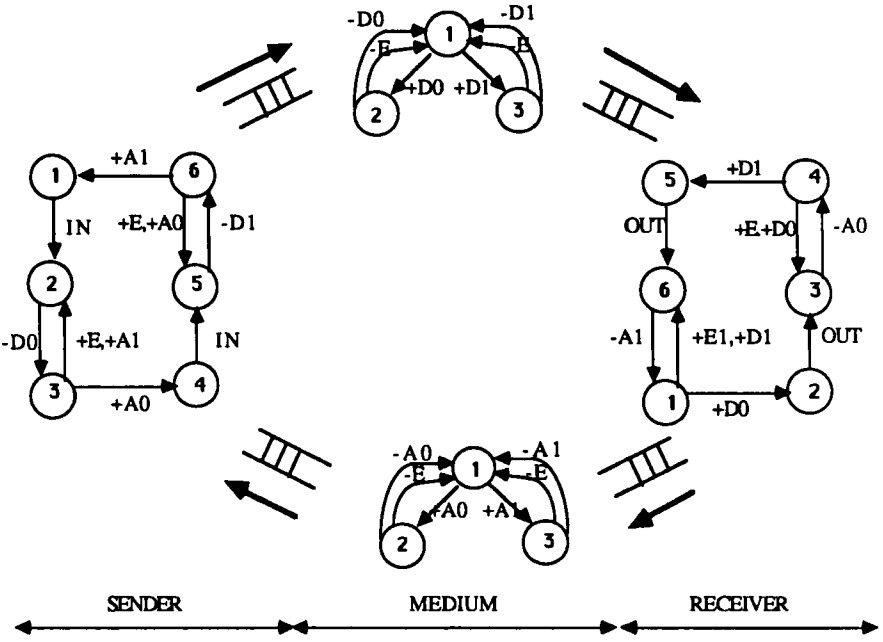


FIG. 5. CFSM model of the alternating bit protocol

data frame is retransmitted until an acknowledgment frame that contains the same alternating bit as the outstanding data frame is received. Retransmission is achieved through the use of an internal timer. The timer is started by the sender upon transmitting a data frame. If no acknowledgment frame is received within a certain predetermined time interval, the sender assumes the transmitted data frame is damaged or lost, and retransmits this data frame. The next data frame will be transmitted only when an acknowledgment of the previous frame has been received before the time expires. The protocol is so called because it uses a single control bit to distinguish between consecutive frames.

The notation used in Fig. 5 is adopted from Bochmann (1978). Labels IN and OUT stand for two service primitives (send and receive, respectively) provided to the user at the higher layer. Event IN receives a data frame from one user on the sender site, while event OUT delivers the received frame to another user on the receiver site. There are two types of data frames (D0 and D1), and two types of acknowledgment frames (A0 and A1). 0 and 1 represent values of the control bit. The + and − signs denote sending and receiving transitions, respectively. For instance, −D0 and +D0 represent transmitting and receiving a data frame with control bit 0, respectively. Transmission errors are shown as E.

Formally, a protocol P in this model is defined as a quadruple

$$P = (\langle Q_i \rangle_{i=1}^{N}, \langle o_i \rangle_{i=1}^{N}, \langle M_{ij} \rangle_{i,j=1}^{N}, \langle \text{succ}_i \rangle_{i=1}^{N}),$$

where

N is the number of protocol entities,

Q_i is the set of state of entity i and $Q_i \cap Q_j = \varnothing$ for $i \neq j$,

o_i represents the initial state of entity i that is an element in Q_i,

M_{ij} represents the messages that can be sent from entity i to entity j and M_{ii} is empty for each i,

succ_i is a partial function mapping for each i and j $(i \neq j)$, $Q_i \times (M_{ij} \cup M_{ji}) \rightarrow Q_j$.

Note that for entity i, $-x$ in the graph denotes that $x \in M_{ij}$ and $+x$ in the graph denotes that $x \in M_{ji}$.

Bochmann (1978) used an FSA model to analyze the ABP and the X.25 call setup and clearing procedures. West and Zafiropulo (1978) used an automated technique to analyze the X.21 and found a number of unspecified reception errors in the 1976 version, which were subsequently corrected in the 1980 version. Gouda and his associates (Gouda and The, 1985; Gouda and Chang, 1984; Gouda and Yu, 1984b) used a network of communicating FSA to model, analyze and synthesize protocols.

Recently, Lee and Lai (1988) have used a relational-algebra approach to represent an FSA as a transition table. On this basis, the well-known theory in relational databases can be used to derive the global-state transitions of the system. Furthermore, the logical errors of protocols can be formulated in terms of relational algebra. This approach has been implemented on the INGRES database system and applied to the validation of several protocols including the X.21.

A limitation of the FSA model is that all necessary information must be represented by explicit states. For example, there must be different states and events to handle each possible sequence number. For complex protocols, the number of states required can be very large, thereby creating the so-called state explosion problem.

3.1.2 Formal Grammars

Formal languages and the grammars that define them are a type of the state-transition model. If one views the sequence of inputs and outputs of an FSA as sentences of a formal language, one can define the formal grammar that would produce all valid sequences. There is a well-known correspondence between such grammars and various types of automata that will recognize (or generate) all valid sequences of the language.

Harangozo (1977) used regular grammars to specify the HDLC protocol and extended the model to handle sequence numbers by indexing the production rules of the grammar. Using context-free grammars, Teng and Liu (1978a, 1978b, 1980) developed a Transmission Grammar (TG) model for the design and implementation of communication protocols.

In the TG model, a protocol is represented by a set of formal grammars. As formal grammars are capable of defining a language, the idea is to come up with a set of production rules that define all the legal protocol action sequences. Each entity or channel of the protocol in the TG model is described by a regular grammar. Production rules in the grammar have the following form:

⟨left-non-terminal⟩ ::= terminal_string ⟨right-non-terminal⟩.

Terminal symbols in the TG production rules represent protocol actions, and non-terminal symbols are equivalent to the states in the FSA model. The meaning of a production rule is that the entity in the state specified by the left-hand non-terminal may take the actions specified by the terminal string and enter the state specified by the right-hand non-terminal.

Terminal actions in the TG model are the following: D (Dequeue), Q (enQueue), F (Fetch), P (Push), O (pOp), C (Clear), E (Equal), N (Non-empty), or U (fUll). The following explanation is obtained from (Teng, 1980).

1. *Queue (Q).* This action inserts the specified message into the specified queue in a First-In-First-Out (FIFO) manner (i.e., it puts the message at the tail of the specified queues). This action requires three fields to be specified. For example, Q.2.msg means inserting (sending) message *msg* to the tail of the queue connected with Entity 2.
2. *Fetch (F).* This action deletes one instance of the specified message from any position in the queue. This action is possible only if at least one instance of the specified message is contained in the queue, and requires three fields to be specified. For example, F.2.msg means fetching message *msg* from any position in the queue connected from Entity 2 to this entity.
3. *Dequeue (D).* This action deletes the specified message from the front of the specified queue. This action is possible only if the specified message is at the front of the specified queue, and requires three fields to be specified. For example, D.2.msg means deleting (receiving) message *msg* from the front of the queue connected with Entity 2 to this entity.
4. *Priority queue (P).* This action inserts the specified message into the specified queue in a Last-In-First-Out (LIFO) manner (i.e., it puts the message at the front of the queue). It is the same as a PUSH operation in a stack structure, and requires three fields to be specified.
5. *Pop (O).* This action deletes the specified message from the end of the specified queue. The action is possible only if the specified message is at the end of the specified queue. All the three fields have to be specified.
6. *Clear (C).* This action deletes all of the messages from the specified input queue, and requires only the first two fields to be specified. For example, C.2 means clearing the queue connected from Entity 2 to this entity.
7. *Empty (E).* This action tests whether the specified ouput queue is empty. This action is possible only if there is no message in the specified queue. Only the first two fields need be specified.
8. *Non-empty (N).* This action tests whether there are messages in the specified output queue. Only the first two fields need be specified.
9. *Full (U).* This action tests whether the number of messages in the specified output queue is equal to its capacity. This action is possible only if the specified queue has reached its limit. Only the first two fields need be specified.

These actions not only enable modeling of a communication medium as FIFO, non-FIFO, and priority queues, but also make status checking of an output queue available. Consequently, they provide a model more powerful than the FSA model, while keeping the model still simple and feasible for automatic verification. As an example, Fig. 6 shows the TG specification of the ABP. The TG model has been automated (see Section 10.6) and used to

```
<1> ::= IN      <2> .
<2> ::= Q.2.D0 <3> .
<3> ::= D.4.A0 <4> ,
        D.4.A1 <2> ,
        D.4.Er  <2> .
<4> ::= IN      <5> .
<5> ::= Q.2.D1 <6> .
<6> ::= D.4.A1 <1> ,
        D.4.A0 <5> ,
        D.4.Er  <5> .
```

```
<IDLE>   ::= D.1.D0 <RECV0> ,
             D.1.D1 <RECV1> .
<RECV0> ::= Q.3.D0 <IDLE> ,
             Q.3.Er <IDLE> .
<RECV1> ::= Q.3.D1 <IDLE> ,
             Q.3.Er <IDLE> .
```

```
<1> ::= D.2.D0 <2> ,
        D.2.D1 <6> ,
        D.2.Er  <6> .
<2> ::= OUT    <3> .
<3> ::= Q.4.A0 <4> .
<4> ::= D.2.D1 <5> ,
        D.2.D0 <3> ,
        D.2.Er  <3> .
<5> ::= OUT    <6> .
<6> ::= Q.4.A1 <1> .
```

```
<IDLE>   ::= D.3.A0 <RECV0> ,
             D.3.A1 <RECV1> .
<RECV0> ::= Q.1.A0 <IDLE> ,
             Q.1.Er <IDLE> .
<RECV1> ::= Q.1.A1 <IDLE> ,
             Q.1.Er <IDLE> .
```

FIG. 6. TG model of the alternating bit protocol

validate the X.21 (Umbaugh and Liu, 1982) and the call setup procedure of the TCP (Umbaugh *et al.*, 1983). It has recently been extended to handle timing constraints (called the TTG model; see Section 7.2).

An extended type of regular expressions (regular grammars), called *protocol expressions*, has been proposed by Holzmann (1982a, 1982b) for the specification and analysis of protocols. Besides the common operators such as union, concatenation and iteration in regular expressions, two new operators are introduced: the division and multiplication operators. The division operator is used to distinguish between input and output actions, whereas the multiplication operator is used to capture the interaction between two protocol expressions. An automated system based on this model has been implemented and will be described in Section 10.4.

Schindler (1980) also extended regular expressions to facilitate protocol specification. The overall expression may be broken into several blocks, each block functioning much as a non-terminal grammar. Each term in the expression may have a rejection predicate that causes an otherwise allowed operation to be deemed invalid if false. Each block may also have several exit blocks, which serve to define alternatives. This model has been used to specify the X.25 (Schindler and Steinacker, 1979; Schindler *et al.*, 1978).

A new methodology, based on attribute grammars, has been proposed by Anderson and Landweber (1984a, 1984b) for specifying and implementing communication protocols. Called Real-Time Asynchronous Grammars (RTAG), it provides mechanisms for specifying data-dependent activities, real-time constraints, and concurrent activities within the protocol entity. To demonstrate the viability of RTAG, a parser has been integrated into the kernel of the 4.2 BSD UNIX operating system, and has been used in conjunction with the RTAG TP-4 specification to obtain an RTAG-based TP-4 implementation in the DoD internet domain.

3.1.3 Petri Nets and Their Derivatives

There is a great deal of research being conducted in the theory and application of Petri nets. During the past 10 years Petri nets have been used to specify and analyze protocols. Recently, Diaz made an extensive survey on this topic (Diaz, 1982). In the Petri nets model, a protocol is modeled by a number of component nets representing different protocol entities. Basically, a Petri net is a graph containing a set of *places* (represented by circles) and a set of *transitions* (represented by bars). Directed arcs are used to connect places to transitions, and transitions to places. A number of tokens distributed in the places represent a marking of the net and also decide which transitions are firable. The firing of a transition causes a redistribution of tokens, and thus moves the net to a new marking. Therefore, places and transitions of a Petri

net specify conditions and events, respectively. How places and transitions are connected can be used to describe the behavior of a protocol.

Formally, a protocol P in this model is defined as a quadruple $\mathbf{P} = (P, T, E, m_0)$, where

P is a finite nonempty set of places,

T is a finite nonempty set of transitions,

E is a set of directed arcs, $E \subseteq P \times T \cup T \times P$, such that for each $t \in T$,

$$(P_i, t) \in E \wedge (t, P_j) \in E, (P_i, P_j \in P)$$

m_0 is an initial marking function that assigns a nonnegative integer number of tokens to each place of the net:

$$m_0 : P \rightarrow \{0, 1, \ldots\}.$$

A transition is defined to be *firable* by a marking m iff every input place of this transition contains at least one token. When a transition is fired, a token is removed from each of its input places and a token is added to each of its output places. This leads the net to a new marking.

For the purpose of presentation, a protocol in this model is often illustrated graphically, as shown by the example of the ABP in Fig. 7. Petri nets have been used to specify and verify the ABP (Merlin, 1976; Postel and Farber, 1976) and the call-setup procedure of a packet-switched network (Symons, 1980) and the ISO Transport protocol (Jurgensen and Vuong, 1984). A variation of the Petri net model, called SARA, has been used at UCLA to model the X.21 (Razouk and Estrin, 1980) and the X.25 (Razouk, 1982).

Pure Petri nets suffer most of the same limitations as FSA. Thus a variety of extensions have been proposed, such as inhibitor arcs, type tokens, and state variables. Another extension is the addition of timing constraints to the transitions (called timed Petri nets) (Berthomieu and Menasche, 1983; Walter, 1983). We will discuss timed Petri nets in Section 7.1.

3.2 Programming Language Models

The programming language model is motivated by the observation that protocols are simply a set of procedures or algorithms to provide communication services. Models falling into this category include abstract programs, temporal logic, and abstract data types. Depending on how high level and abstract a language is used, this approach to specification may be quite near to an implementation of the protocol. However, efforts to prove the correctness of the program (the *safety* and *liveness* properties) far exceed those required for developing the program, and its correctness proof usually depends heavily on human ingenuity and is hard to automate.

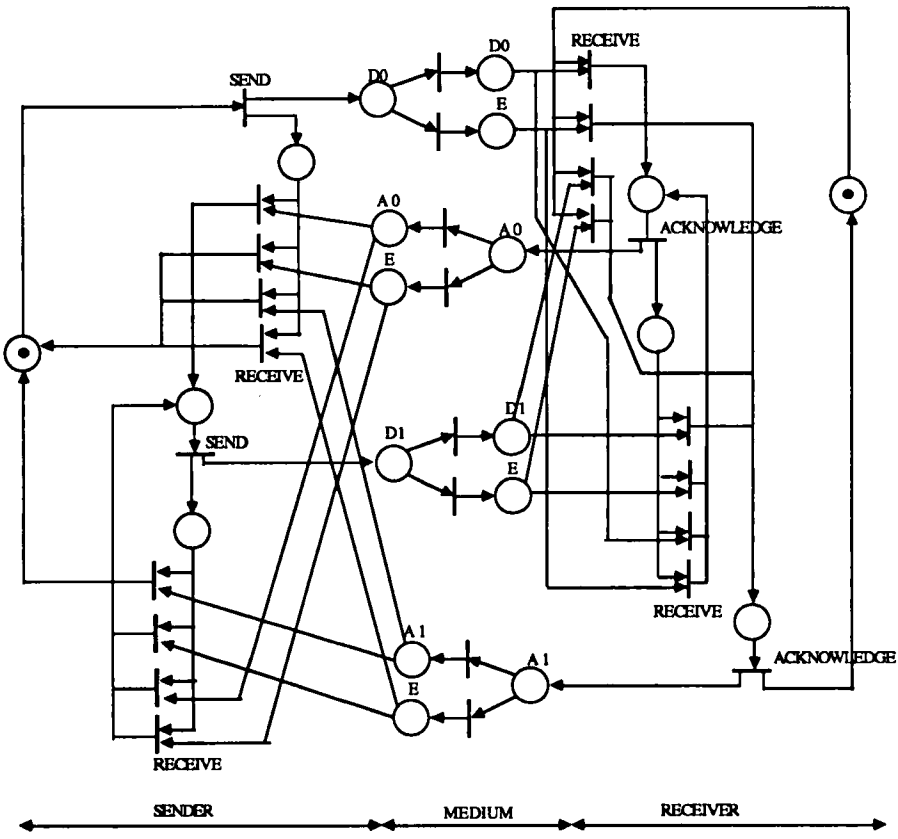


FIG. 7. Petri net model of the alternating bit protocol

3.2.1 *Abstract Programs*

The use of programming languages for specifying communication protocols is motivated by the observation that protocols are simply one kind of algorithm, and that programming languages provide a clear and concise way of describing algorithms. In an abstract program model, protocols are described as parallel programs. Figure 8 shows an abstract program model for specifying a simple protocol called the Alternating Bit Protocol (ABP).

Bochman (1975) made one of the earliest attempts at specifying and verifying a simple HDLC protocol using an abstract program. The protocol was specified in a free-style Pascal. The program structure is event driven and similar to a state-transition model in many ways. He partially verified the protocol by stating three safety invariants that described the number of

```
Clock

1:  TIME := TIME -1
2:  delay
3:  goto 1

Medium

1:  wait SEND                  /* wait until Sender ready to send */
2:  SEND := false              /* turn off indicator */
3:  LOSS := ?                  /* decide whether message should
                                  be lost */
4:  if LOSS then goto 1        /* quit sending message */
5:  ERROR := ?                 /* decide whether error should
                                  happen */
6:  if ERROR
7:     then SEQNB := error     /* transmit indication of error */
8:     else betgin
             DATA_RECEIVED := DATA_SENT
             SEQNB := SEQ
          end                  /* normal transmission */
9:  wait RECEIVE               /* wait for acknowledgment */
10: RECEIVE := false           /* turn off indicator */
11: LOSS := ?                  /* decide whether ack should get
                                  lost */
12: if LOSS then goto 1        /* quit sending acknowledgment */
13: ERROR := ?                 /* decide whether ack should
                                  be erroneous */
14: if ERROR
15:    then ACK := error       /* transmit indication of error */
16:    else  ACK := EXP        /* normal acknowledgment */
17: goto 1                     /* repeat for next message */
```

FIG. 8. Abstract program of the alternating bit protocol

messages sent and received by each protocol entity. However, the proof was not formal.

About the same time Stenning (1976) also used an abstract program to specify and verify a data-transfer protocol. His code was very close to standard Pascal, which enabled him to rely on the standard Pascal rules for deriving pre- and post-conditions of the invariant assertions. Using the Floyd-Hoare technique (Floyd, 1967; Hoare, 1969), he was able to verify the safety property of his protocol.

Krogdahl (1978) developed the technique of *protocol skeletons* for specifying and verifying safety properties of classes of protocols. He attempted to

Sender

```
1:  DATA_SENT := INDATA[PT]   /* get next message to be sent */
2:  PT := PT + 1              /* prepare PT for next message */
3:  SEQ := SEQ + 1 MODULO 2   /* switch sequencer for next msg */
4:  ACK := none               /* erase previous ack */
5:  TIME := TO                /* initialize timer to timeout
                                 interval */
6:  SEND := true              /* send message */
7:  wait (ACK ≠ none or TIME = 0)/* wait for ack or timeout */
8:  if ACK = SEQ then goto 1  /* O.K., repeat */
9:     else goto 4            /* error or timeout, try again */
```

Receiver

```
1:  wait SEQNB ≠ none         /* wait for a message */
2:  if (SEQNB = error or SEQNB ≠ EXP)
3:     then goto 5            /* send old ack for message */
4:     else begin
              OUTDATA := DATA_RECEIVED
               EXP := EXP + 1 MODULO 2
            end               /* append received message and
                                 prepare ack */
5:  SEQNB := none             /* cancel indicator */
6:  RECEIVE := true           /* send ack*/
7:  goto 1                    /* repeat for next message */
```

FIG. 8. (*continued*)

provide as general a program specification as possible, using an Algol-like language. The proof of the invariants follows the standard Floyd-Hoare technique.

Ansart *et al.* (1982) developed a Protocol Description and Implementation Language (PDIL) for specifying protocols and allowing automatic implementation. Based on standard Pascal, PDIL relieves the user of all the constraints of putting into a programming language form (e.g., the definition of data structures and procedures for manipulating typed objects). The latter work is done by a preprocessor for PDIL, which generates coherent Pascal text.

Castanet *et al.* (1985) presented a methodology of using Ada for the specification and implementation of protocols. Compared with other programming languages, Ada has the advantage of homogeneity; its main drawback is in performance. Yelowitz *et al.* (1982) combined the use of Ada and AFFIRM (Gerhart *et al.*, 1980) for modeling a fiber-optic token-ring network.

3.2.2 CSP and CCS

Two of the abstract program models that have been receiving considerable attention in the literature are Hoare's Communicating Sequential Processes (CSP) and Milner's Calculus of Communicating Systems (CCS). CSP (Hoare, 1978) is a high-level concurrent language designed for distributed systems. A CSP program consists of a number of processes that are mutually disjoint in address space, and communications between processes are accomplished only through message passing. In addition, guarded commands are used to describe nondeterministic behavior of each process. A protocol in this model is thus represented by a CSP program, in which each protocol entity is represented by a process.

Major CSP constructs are described briefly as follows:

1. Parallel commands $[P_1 \| P_2 \| \ldots \| P_n]$ specify the concurrent execution of n processes $P_1, P_2, \ldots, P_n$.
2. Input command $P_j?(x)$ and output command $P_i!(expression)$ specify the communication between processes P_i and P_j. (Process P_j sends the value of *expression* to variable x of process P_i.)
3. Both alternative command

$$
\begin{array}{l}
[\\
b_1;\ I/O_1 \rightarrow command\ list_1 | \\
b_2;\ I/O_2 \rightarrow command\ list_2 | \\
\vdots \\
b_n;\ I/O_n \rightarrow command\ list_n \\
]
\end{array}
$$

and repetitive command

$$
\begin{array}{l}
*[\\
b_1;\ I/O_1 \rightarrow command\ list_1 | \\
b_2;\ I/O_2 \rightarrow command\ list_2 | \\
\vdots \\
b_n;\ I/O_n \rightarrow command\ list_n \\
]
\end{array}
$$

are in the form of guarded commands and can be used to specify nondeterministic behavior of a protocol.

As an example, a CSP specification of the ABP is shown in Fig. 9.

```
ABP :: [ Sender || Medium || Receiver]

Sender ::
frame : record
          data : ...;
          seq : (0,1,error)
        end;
DATA : ...; SEQ : (0,1);
Ack : (0,1,error); done : boolean;
SEQ := 1;
*[User1?(DATA) → SEQ := (SEQ+1) mod 2;
                 frame.data := DATA;
                 frame.seq := SEQ;
                 done := false;
                 *[¬done; Medium!(frame) → Medium?(Ack);
                                           [ Ack = SEQ → done := true |
                                             Ack = (SEQ+1) mod 2 → skip |
                                             Ack = error → skip
                                           ]
                 ]
]

Receiver ::
frame : /* same as in Sender */
exp : (0,1);
exp := 1;
*[Medium?(frame) → [ frame.seq = (exp+1) mod 2 → User2!(frame.data);
                                                 exp := (exp+1) mod2 |
                     frame.seq = exp → skip |
                     frame.seq = error → skip
                   ];
                   Medium!(exp)
]

Medium ::
frame : /* same as in Sender */
Ack : (0,1,error);
correct, corrupted : boolean;
correct := true; corrupted := true;
*[Sender?(frame) → [correct → Receiver!(frame) |
                    corrupted → frame.seq:=error;
                                Receiver!(frame)
                   ] |
  Receiver?(Ack) → [correct → Sender!(Ack) |
                    corrupted → Sender!(error)
                   ]
]
```

FIG. 9. CSP model of the alternating bit protocol

On the other hand, CCS (Milner, 1980) is a language for specifying the communication behavior of concurrent systems in terms of a small set of operators. In this model, a protocol is represented by a set of communicating agents. An agent is capable of communicating with other agents (via internal ports) or with an external observer of the system (via external ports). The basic notion in CCS is a set of atomic events denoting either internal events or communication events. These atomic events are represented as follows:

1. αx for input event, where x is a *value variable.*
2. $\bar{\alpha} e$ for output event, where $\bar{\alpha}$ is a label complementary to α, and e is a *value expression.*
3. τ for internal event.

Based on the notion of the occurrence of an event, CCS has operators to express the following:

1. Sequences of events, by operator ".".
2. Choice between sequences of events, by operator "+".
3. Recursion for specifying infinite sequences, by operator "⇐".
4. Parallel composition of agents to form systems of communicating agents, by operator "||".
5. Hiding of a subset of the internal ports, enabling one to abstract away from the internal details of an agent, by operator "\".

Formally, a protocol in CCS is defined by the following BNF notation:

$$t ::= x | op(t_1, t_2, \ldots, t_n) | x \Leftarrow t$$

where x is a variable name, op is an operator, and $t\ (t_1, t_2, \ldots, t_n)$ is a CCS expression.

As an example, a CCS specification of the ABP is shown in Fig. 10. Note that in this example, α and β are data flowing from Sender to Receiver, δ and γ are acknowledgments flowing from Receiver to Sender, and I and O are communication actions with outside observers.

The global behavior of a protocol in the CCS model can be computed by applying the operation of parallel composition to all its communicating agents. For example, four communicating agents of the above ABP bit protocol can be composed as follows:

$$(S \| C_1 \| R_0 \| C_2).$$

During the parallel composition, pairs of events such as αx and $\bar{\alpha} e$ can be coupled and become a rendezvous event $x := e$. The global behavior of the protocol can again be represented by a CCS expression using only operators ".", "+", and "⇐".

Sender:

$$S \Leftarrow I.S_0$$
$$S_0 \Leftarrow \bar{\alpha}_0.\{\delta_0.I.S_1 + \delta_1.S_0 + \delta_e.S_0\}$$
$$S_1 \Leftarrow \bar{\alpha}_1.\{\delta_1.I.S_0 + \delta_0.S_1 + \delta_e.S_1\}$$

Receiver:

$$R_0 \Leftarrow \beta_0.\bar{O}.\bar{\gamma}_0.R_1 + \beta_1.\bar{\gamma}_1.R_0 + \beta_e.\bar{\gamma}_1.R_0$$
$$R_1 \Leftarrow \beta_0.\bar{\gamma}_0.R_1 + \beta_1.\bar{O}.\bar{\gamma}_1.R_0 + \beta_e.\bar{\gamma}_0.R_1$$

Communication Medium:

$$C_1 \Leftarrow \alpha_0.\{\bar{\beta}_0.C_1 + \bar{\beta}_e.C_1\} + \alpha_1.\{\bar{\beta}_1.C_1 + \bar{\beta}_e.C_1\}$$
$$C_2 \Leftarrow \gamma_0.\{\bar{\delta}_0.C_2 + \bar{\delta}_e.C_2\} + \gamma_1.\{\bar{\delta}_1.C_2 + \bar{\delta}_e.C_2\}$$

FIG. 10. CCS model of the alternating bit protocol

Recently, Liu and Liu (1984, 1986) proposed a methodology for specifying and analyzing protocols and services for conformity analysis. They specified both a protocol and its service by a CSP based language. To perform the conformity analysis, they developed a transformational system to extract from a CSP process the communication sequences that may arise during its execution and to express these sequences in terms of behavior expressions in CCS. By performing algebraic manipulations and the equivalence proof on these expressions, they can show that the external behavior of a protocol conforms to its intended services. A version of the ABP was used to demonstrate the feasibility of this methodology. We will return to this topic in Section 5.2.

3.2.3 *Temporal Logic Techniques*

Temporal logic was first introduced by Pnueli (1977) as an adaptation of a classical model logic suitable for defining the semantics of computer programs. Recently, it has been used by Hailpern and Owicki (1980) and Schwartz and Melliar-Smith (1981) to specify and verify the liveness (or progress) property of protocols. The liveness property requires that certain transitions eventually take place, and is difficult or impossible to state and prove in state-transition specifications, since conventional logic cannot refer to any state other than the present one.

Hailpern and Owicki (1980) model a protocol system as a set of interacting modules that represent the logical units of the system. Both active (called process) and passive (called monitor) modules may be specified. They exploit this modularity in their specifications and proofs. They have verified the safety

and liveness properties of the ABP, and Stenning's data-transfer protocol (Hailpern and Owicki, 1983).

Schwartz and Melliar-Smith (1981) developed specifications employing temporal logic with a more explicit notion of system state. They divide the task of specifying and verifying protocols into two parts: service-level specification and network-level specification. The service level defines the operations available to the users of the protocol, while the network level represents an abstract specification of the essential details of the protocol implementation. The goal is to verify the service level from the network level and to verify the network level from the protocol code. They illustrated the feasibility of their technique by formally verifying both safety and liveness properties of the ABP.

Recently, Sabnani and Schwartz (1984) verified a multidestination protocol, called the Selective Repeat procedure, for a satellite broadcast channel shared by using a time-division multiplexed technique. The Selective Repeat procedure is modeled as a parallel program in a Pascal-like language. Sabnani and Schwartz show the correctness of the parallel program model using temporal logic so that both the safety and liveness properties are satisfied.

3.2.4 Abstract Data Types

Abstract data types (Guttag, 1975) are an attempt to encapsulate data and the operations that manipulate it. There are two approaches in this area: abstract model and axiomatic. However, the distinction between these two approaches may not be so great in practice, since it is possible to write abstract model specifications in the axiomatic notation.

As reported by Sunshine (1982b, 1983), experience with these techniques is still limited. However, due to its ability to formalize a large class of protocols, coupled with the existence of some automated tools for checking specifications, the abstract data type approach to protocol design looks very promising. Thus, much more research is required in this direction.

The major advantage of programming language models over state-transition models is their capability to handle variables and parameters, such as sequence numbers and timers, which may take on values of wide range. Another advantage is their ability to specify all protocols and most of their properties rather than only general correctness properties.

However, since protocol specifications in programming language models may be very similar to actual implementations, unessential features are often combined with the essential algorithms. In addition, efforts to prove the correctness of programs representing communication protocols far exceed those required to develop algorithms or programs. Program proof usually depends heavily on human ingenuity and intuition, and the automation of proof steps seems quite impossible and, therefore, is still far away from being of significant use.

3.3 Hybrid Models

The hybrid model attempts to combine the advantages of both state-transition and programming language models. It typically uses a small number of states to capture only the main features of the protocol, with each state being augmented with context variables and processing routines. The state-transition part of the model captures the control aspects of the protocol while variables and data are easily handled by the program part of the model. Recently, hybride models seem to be receiving the most attention, and both the ISO and the CCITT are actively developing standard techniques based on a hybrid model.

3.3.1 Abstract Machines

The abstract or extended finite-state machine (EFSM) model is a generalization of the FSA model. The abstract model allows multiple-state variables of various types; the *state* now becomes a vector of these variables and the transition functions become more complex. The values of these state variables are changed by the occurrence of events. An event can occur only if certain enabling conditions are satisfied. (An enabling condition is a predicate on the state variables.) When more than one event in an event-driven system is enabled, any one of the enabled events is allowed to occur, resulting in nondeterminism.

In the abstract machine model, each protocol entity P_i is represented by a vector of state variables V_i. Each state variable $v_j \in V_i$ can take on values from a domain D_j. One of these state variables can be regarded as an explicit state variable. A channel between two entities, C_k, is represented by state variable z_k, which is the message sequence contained in the channel. Thus, the global state of the protocol system is given by the tuple $(V_1, \ldots, V_n; z_1, \ldots, z_m)$. The initial global state of the system is given by the initial value of each state variable, and all communication channels are initially empty. The values of these state variables are changed by the occurrence of *events*. An event is described by a predicate that relates the values of the state variables immediately before the event occurrence to their values immediately after the event occurrence. Thus it is denoted by predicate $pred(V; V'')$ or $pred(V, z, \ldots; V'', z'', \ldots)$, where V and z are variables before event occurrence and V'' and z'' are variables after event occurrence. The predicate embodies specifications of both the event's enabling conditions and actions.

Each entity or channel has a set of events. The events of entity P_i can only involve the state vector V_i and the state vectors of channels accessible from P_i. Entity events model message receptions, message sends, and internal activities. The events of channel C_k can involve only the state vector z_k. Channel events model channel errors such as loss of messages in transit. An event can occur

Variables of Sender:

state:(1,2,3); explicit state variable of sender.
seq:(0,1); sequence number of message sent.
ack:(0,1,error); acknowledgment from receiver.
data:. . .; data to be transferred.

Events of Senders:

1. AcceptData(V_1;V_1'') ==
   state = 1 **and** In(data) **and** seq := seq + 1 (mod 2) **and** state := 2;

2. SendData(V_1,z_1;V_1'',z_1'') ==
   state = 2 **and** $Send_1$((seq,data),z_1;z_1'') **and** state := 3;

3. ReceiveAck(V_1,z_2;V_1'',z_2'') ==
   state = 3 **and** $Receive_2$(z_2;(ack),z_2'') **and**
   ((ack = seq) → state := 1
   | (ack = seq + 1 (mod 2)) → state := 2
   | (ack = error) → state := 2
)

Variables of Receiver:

state:(1,2,3); explicit state variable of receiver.
exp:(0,1); opposite of expected sequence number of message received.
seqnb:(0,1,error); sequence number of received message.
data:. . .; data in received message.

Events of Receiver:

1. DeliverData(V_2;V_2'') ==
   state = 1 **and** Out(data) **and** exp := exp + 1 (mod 2) **and** state := 2;

2. SendAck(V_2,z_2;V_2'',z_2'') == state = 2 **and** $Send_2$((exp),z_2;z_2'') **and** state := 3;

3. ReceiveData(V_2,z_1;V_2'',z_1'') ==
   state = 3 **and** $Receive_1$(z_1;(seqnb,data),z_1'') **and**
   ((seqnb = exp + 1 (mod 2)) → state := 1
   | (seqnb = exp) → state := 2
   | (seqnb = error) → state := 2
)

Event of Medium from Sender to Receiver:

ChannelError(z_1;z_1'') == (for some (seq,data) in z_1) [seq'' := error]

Event of Medium from Receiver to Sender:

ChannelError(z_2;z_2'') == (for some (seq) in z_2) [seq'' := error]

FIG. 11. Abstract machine model of the alternating bit protocol

only if its enabling conditions are satisfied. When more than one event in such an event-driven system are enabled, any one of the enabled events is allowed to occur. An entity event can access channel state variables only via send and receive primitives. The send primitive for channel C_k is defined by $\text{Send}_k(z_k, m; z_k'') = (z_k'' = (z_k, m))$. The receive primitive for channel C_k is defined by $\text{Receive}_k(z_k; m, z_k'') = ((m, z_k'') = z_k)$.

As an example, the abstract machine specification of the ABP is shown in Fig. 11. Many researchers have proposed particular forms of such abstract machine models for specifying protocols. While differing in names and in the details of syntax, they may be considered equivalent in expressive power.

Based on Kelley's transition model for parallel programs (Keller, 1976), Bochmann (1980) has proposed a general transition model for the formal specification of protocols, the specification of the services provided, and the verification of the correct operation. He discussed these issues by considering, as an example, the HDLC classes of procedures.

Recently, we have extended the TG model, called the Extended Transmission Grammar (ETG) model, to contain context variables such as sequence numbers in protocol specifications (Chu, 1989). We borrow the notion of Communicating Sequential Processes (Hoare, 1978) in using "?" and "!" as the "receive" and "send" events, respectively. In this context, "?" is a blocking "receive" as it is in CSP; whereas "!" is a non-blocking "send" that will not wait for the communicating partner to be ready for the "send" event to be executable, as is required for "!" in CSP. As an example, the specification of the ABP in the ETG model is shown in Fig. 12. An automated validation package for the ETG model has been developed (see Section 10.6).

Shankar and Lam (1983) used an event-driven process model to specify a version of the HDLC protocol between two communicating protocol entities. The protocol is verified using the method of *projections* (Lam and Shankar, 1984). The verification serves as a rigorous exercise to demonstrate the applicability of this method to the analysis of realistic protocols. The procedure has been automated and is described further in Section 10.2.

3.3.2 *Estelle and LOTOS*

Early in the development of the OSI Reference Model (see Fig. 2), it was recognized that formal description techniques (FDT) would be required to accomplish the goals of OSI. Work has been under way within the ISO to use FDT for writing precise specifications for the OSI protocols. One of the FDT developed by Subgroup B, called ESTL (Extended State Transition Language) or "Estelle," is a hybrid model based on Pascal and an FSA model (Bochmann 1981; Linn, 1985; Tenney, 1983; Vissers *et al.*, 1983). Experience with the use of this FDT for communication services and protocols is reported in (Bochmann

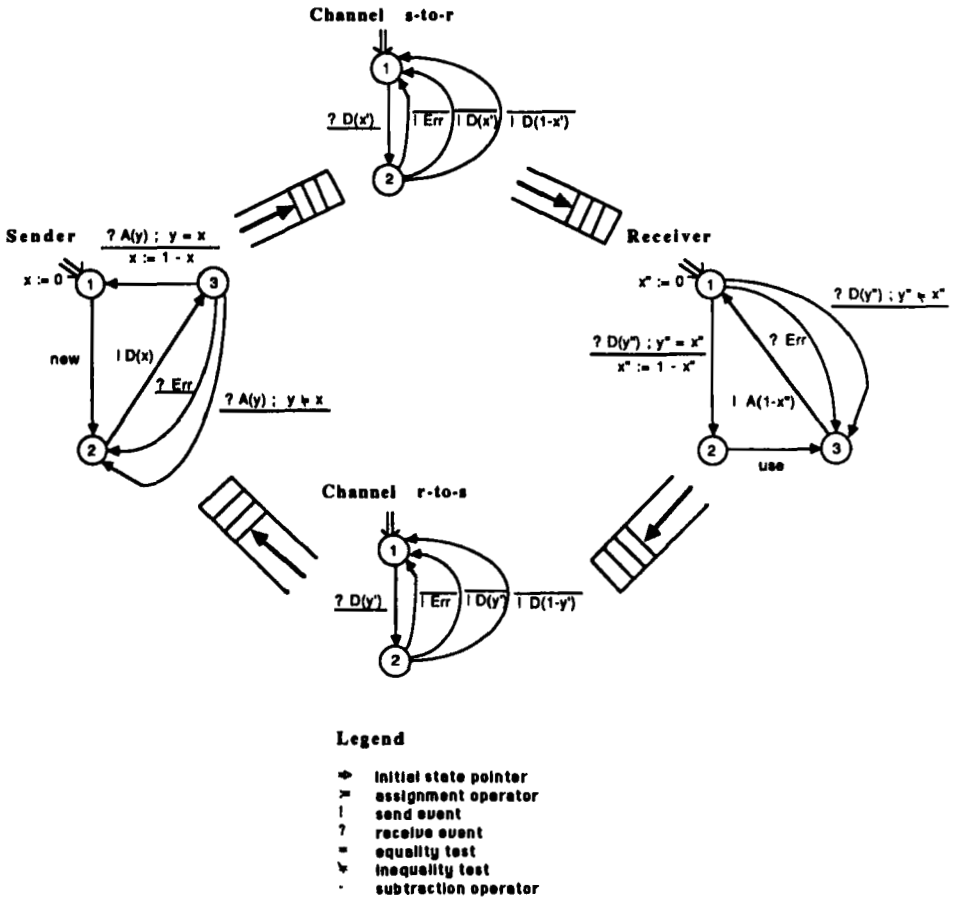


FIG. 12. ETG model of the alternating bit protocol

et al., 1982a, 1982b). A tutorial on Estelle can be found in (Budkowski and Dembinski, 1987). Since September 1988, Estelle has become an international standard, IS 9074 (Diaz et al., 1989).

The language being developed by the FTD Subgroup C is called LOTOS (Language for Temporal Ordering Specification). Based on Milner's Calculus of Communicating Systems (Milner, 1980), it aims to assist in the formal definition of protocols and services for the OSI Reference Model. The great promise of LOTOS lies in the fact that it allows as many levels of refinement as are needed, through the use of two language operators: parallel composition and restriction. However, since the effort is made only to describe the message sequences, there is a minimum impact on the specification of an implementation, thereby giving the implementor the maximum amount of

freedom yet still providing sufficient guidance to ensure compatibility. Recently, Brinksma has used LOTOS to specify the OSI transport service (Brinksma and Karjoth, 1984). A tutorial on LOTOS can be found in (Bolognesi and Brinksma, 1987). By now it has become an international standard, IS 8807 (van Eijk et al., 1989).

3.3.3 SDL

Since 1968, the International Telegraph and Telephone Consultative Committee (CCITT) has made an effort to create a new language to precisely specify and describe the functional features of a system. The resulting language is called Specification and Description Language (SDL), which is also an extended FSM language (Dickson and deChazal, 1983; Rockstrom and Saracco, 1982). SDL has both procedural and declarative constructs which together provide expressive and powerful means for modeling specifications. It is widely used in telecommunication applications and is supported by numerous tools. A refinement of SDL in a Pascal-oriented language is under consideration by the CCITT. A tutorial on SDL can be found in (Saracco and Tilanus, 1987).

3.3.4 FAPL

The language IBM uses for describing SNA is called Format and Protocol Language (FAPL). It is derived from PL/1 and contains additional constructs for handling FSMs and processes. A protocol specified in this form is precise, readily accessible to the implementing product designer and programmers, and structurally close to the implementations (Nash, 1983; Pozefsky and Smith, 1982). A tutorial on FAPL can be found in (Nash, 1987).

3.3.5 Selection/Resolution Model

Aggarwal *et al.* (1983) has proposed the Selection/Resolution model for specifying, analyzing and validating the behavior of protocols. The model centers around abstract entities called processes. Parallelism is addresed directly with concurrent transitions in each process dependent on the status (formally defined selections) of neighboring process. Protocol specification is accomplished by defining many small interacting processes, each easy to specify, which collectively describe the behavior of the protocol. The model is based on an abstract calculus, which is amenable to hierachical specification. Validation of a specification is precisely defined in terms of proving properties on the trajectories of processes. The feasibility of the model was demonstrated by applying it to the ABP and a file-transfer protocol (Aggarwal and Sabnani, 1986).

3.3.6 CIL

Krumm and Drobnik (1983, 1984) proposed the CIL (Communication Service Implementation Language) approach for the development of communication services. It is based on an event-oriented model of program execution and a first-order predicate calculus. The verification of a program written in CIL is supported by the automated generation of program axioms and by the predicate calculus. The design of a program realizing a transport service exemplifies the CIL approach.

4. Protocol Validation

As discussed in the previous section, to ensure that a communication system functions properly, its communication protocols must be specified unambiguously so that the protocols can be implemented faithfully. More important, the protocols must be shown to be correct. Verification or validation is the process of showing the correctness of a protocol. Verification and validation are often used interchangeably. We will follow the terminology used by Sunshine (1979). That is, protocol verification is a demonstration that the interactions of the communicating entities, based on their protocol specification and the specification of the services provided by the layer below, satisfy the service specification, whereas protocol validation refers to the more limited analysis that the protocol specification satisfies a number of general correctness properties that are essential to all, or nearly all, protocols. The list of general correctness properties that must be satisfied by all protocols is as follows:

1. *Completeness.* The protocol must be able to handle all conditions that may arise.
2. *Freedom from Deadlock.* Each protocol system or global state allows for progress to some other state.
3. *Absence of Tempo-Blocking Loops.* All looping paths provide some meaningful communication operations.
4. *Freedom from Livelock.* Tempo-blocking loops, if any, provide some exit to paths along which meaningful communication operations may take place.
5. *Freedom from Overflow.* The protocol is not allowed to place more messages than the communication channels can accommodate.
6. *Termination.* The protocol will arrive at the desired final situation.

As can be seen from the discussions in the previous section, approaches to protocol validation depend heavily on the models used for specification, and

they have followed two main paths: *reachability analysis* and *deductive inference* (or program proofs). Reachability analysis is based on exhaustively exploring all the possible interactions of the communicating protocol entities within a layer, whereas deductive inference is based on a list of statements of properties (safety and liveness) and a list of axioms and rules for inferring the statements from the axioms. Restricting our discussion in this section to reachability analysis only, we will survey relief strategies proposed by a number of researchers to deal with the so-called state explosion problem, and also propose a novel approach of our own, which is based on the search strategies developed in the field of Artificial Intelligence (AI).

4.1 Reachability Analysis

Reachability analysis has been proved to be one of the most effective ways for analyzing state-oriented models of communication protocols. It was first proposed by West (1978a, 1978b) and later improved by a number of researchers (see Section 4.2). The method is based on the idea of *state perturbation* in which all the possible global states of a protocol are enumerated from an initial state. Properties of the protocol can then be verified based on the global states and the global state reachability graph.

Validation techniques used by FSA models are all based on some sort of reachability analysis. This analysis involves the exploration of all possible interactions among communicating entities. A *global, system,* or *composite state* is a combination of both the states of communicating entities and the states of communication media. From the initial global state, new global states are generated by applying all possible transitions (user commands, message arrivals, internal timeouts). This process continues for each newly generated global state until no new states can be generated. The resulting graph is called the *reachability graph.*

Reachability analysis is well suited to checking the general correctness properties described above since these properties are a direct consequence of the structure of the reachability graph. For example, global states with no exits are either deadlock states or proper termination states. More importantly the generation of the global state space can be easily automated and several automated systems for protocol validation have been developed (see Section 10).

A global state graph of the above ABP is shown in Fig. 13. This graph is generated under the assumption of so-called *empty medium abstraction* (Bochmann, 1978). Under this assumption, communication media are considered empty, i.e., no message is in transit. Therefore, a global state is composed of the states of the communicating entities. In this graph, (x, y) represents a global state where the sender is in state x and the receiver in state y.

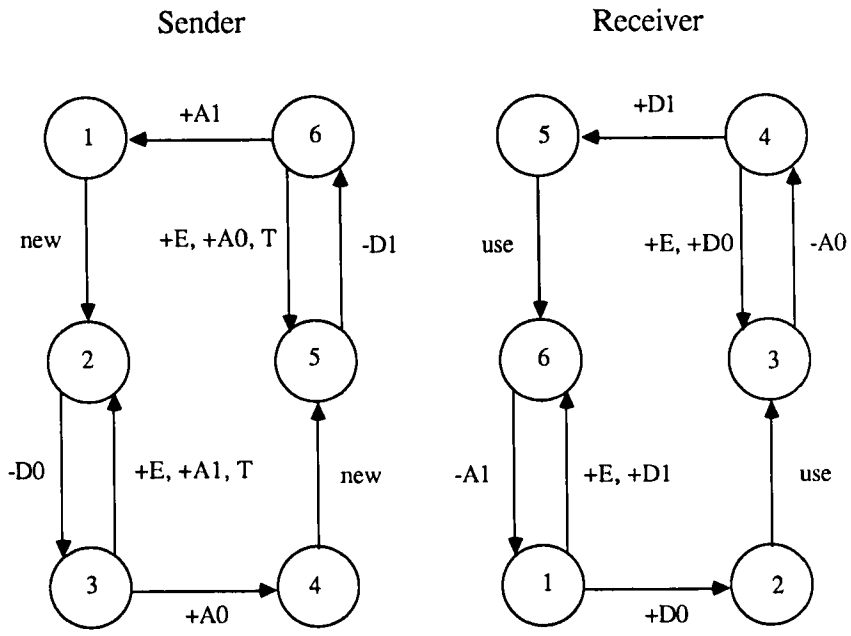


FIG. 13A. FSA model of the alternating bit protocol (with Timeout Mechanism)

A possible transition consists of the sending of a message by one entity, and the reception of this message (or erroneous message) by the other entity. In the case of message loss, a transition corresponds to only the sending of a message. The transition labeled D, stands for reliable transmission (followed by reception) of a data frame with control bit i, where i is 0 or 1. Di^E shows that the data frame is damaged on transmission, and the erroneous frame is received by the receiver. Di^L represents that the data frame is lost on transmission, and no data frame is received by the receiver. The same notation is used for the acknowledgment frame except that D has been changed to A.

The resulting global state graph may be examined for detecting general correctness properties. For example, in Fig. 13B, each global state can go back to the initial global state, thus indicating the absence of deadlocks. There exist loops without progress (or livelocked loops) such as the loop consisting of nodes 2 and 10, 4 and 12, 6 and 15, and 8 and 13. These loops are executed in the case of transmission errors or losses, and may be prevented by setting a limit to the number of retransmission times.

The main advantage of FSA models is that reachability exploration can be automated. The process of validation is far too time consuming and error prone if done by hand. It may be possible to carry out validation on simple

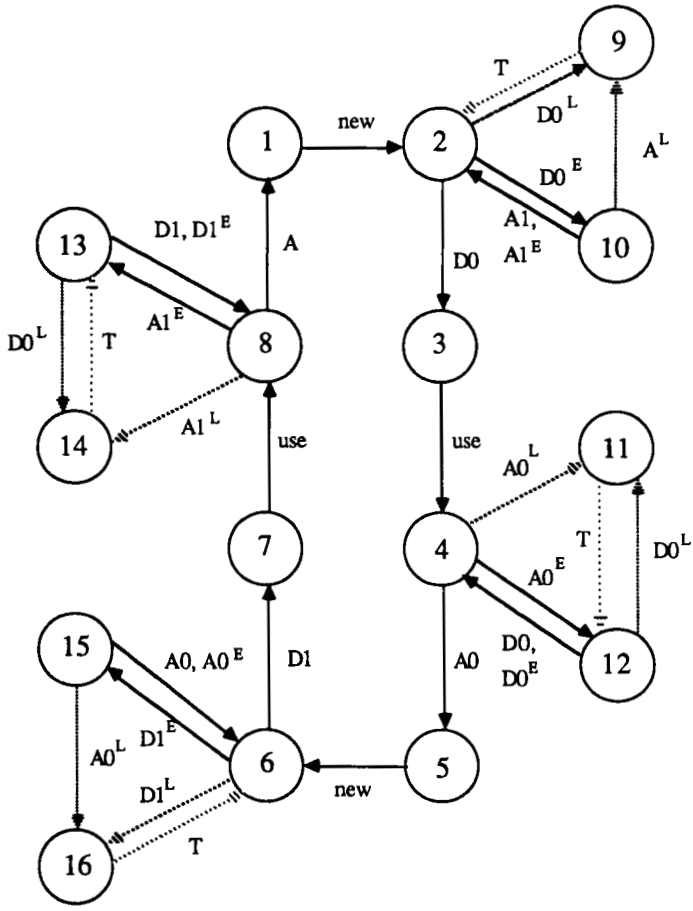


FIG. 13B. Global state graph of the alternating bit protocol under the empty medium abstraction

protocols by hand. As protocols become more and more complex, the effort of manual validation grows beyond human capability. With the help of an automated validation program, tremendous design time consumed by the protocol designer can be saved.

The major difficulty of such models is *state space explosion* (the size of the global state graph grows exponentially with protocol complexity). For complex protocols, this technique becomes too complicated for a complete generation and examination of all reachable global states. Thus, state-transition approaches are not all suitable for modeling variables that may take on a large number of values.

4.2 Relief Strategies

Due to its effectiveness and ease of mechanization as discussed above, many protocol validation tools have been built based on the method of reachability analysis. However, the applicability of this method is severely restricted by the so-called *state space explosion* problem. Many researchers have developed *relief strategies* to attack the state space explosion problem. In this section a brief survey of these strategies is presented.

The relief strategies presented in this section can be classified into three categories according to when they should be applied. The strategies in the first category are those applied during protocol modeling, i.e., in the stage of formally specifying protocols. The second category of relief strategies are applied after the protocol modeling is done but before the actual validation is performed. The third category of strategies are those incorporated into the validation (and thus reachability analysis) algorithms.

1. The relief strategy proposed by West (1982) falls into the first category. The major techniques proposed by him are as follow:

(a) Restricting the use of many-valued parameters such as sequence numbers in the specification.
(b) Limiting the number of messages underway in the message queues.
(c) Limiting the classes of transmission errors under consideration.

2. Though different terms such as decomposition (Vuong and Cowan, 1982a; Choi and Miller, 1983), and multi-phase (Chow, 1985) are used by these groups of researchers, the relief strategies they proposed basically follow the same direction. They observe that certain classes of protocols can be decomposed into components (or multiple phases), which then can be separately verified to ensure the correctness of the original protocol. This reduces the complexity of the verification problem since protocol components are always smaller in the numbers of states and transitions than the original protocol. They are relief strategies of the second category as classified at the beginning of this section.

3. The strategy proposed by Lam and Shankar (1984) also belongs to the second category. Unlike the strategies of decomposition, Lam and Shankar proposed the *projection* approach, which, instead of partitioning a protocol into multiple phases, constructs from the given protocol an image protocol for each of the functions that is intended to be verified. The states, messages, and events of entities in an image protocol are obtained by aggregating groups of states, messages, and events of corresponding entities in the original protocol. The resulting protocol is smaller than the original protocol, and therefore the complexity of the problem is reduced.

The following relief strategies all belong to the third category.

4. The Finite State Machine (FSM) analyzer, built as one of the tools in the protocol development system by Blumer and Sidhu (1986), is based on the model of the extended finite state machine (see Section 10.5). A mechanism called *transition choice rule* is provided, which is associated with each of the transitions. The choice rule is a Boolean condition whose value decides whether or not the associated transition of the FSM is to be executed during the reachability analysis. For example, a rule may specify that no transition may be executed twice in the same path, or that no transition may be executed to bring a state to itself. As a result, the scope of the state exploration is controlled by the choice rules. For instance, infinite loops that may occur in the analysis can be eliminated with appropriate choice rules.

5. LISE (Ansart, 1985) is also a tool based on the model of the extended finite state machine. It can be operated in two modes: *validation mode* and *simulation mode.* When the system is operated in validation mode, it fires all the possible transitions in every global state. On the other hand, if the system is in simulation mode, only one transition out of a global state is selected to fire. The simulation mode is adopted whenever it turns out that a complete validation is infeasible due to state explosion. Selection in simulation mode is accomplished in two ways. In the first way, the selection is simply done on a random basis; in the second way, a priority is assigned to each of the transitions and the transition with the highest priority is always the one chosen.

6. This group of strategies (Rudin and West, 1982; Gouda and Han, 1985; Zhao and Bochmann, 1986) are based on the *fair progress state exploration.* This was first proposed by Rudin and West (1982), then extended by other researchers. The idea is to explore only those global states that are reachable, provided that two protocol entities proceed at the same speed. Protocol design errors such as deadlocks and unspecified receptions can still be completely detected though the exploration is not exhaustive. The limitation of this strategy is that it only applies to two-entity protocols.

7. This strategy is called the *maximal progress state exploration* (Gouda and Yu, 1984b). The idea is similar to that of the fair progress state exploration and its applicability is also limited to two-entity protocols. Basically, the strategy is that the global states of a two-entity protocol can be generated in two separate explorations, during each of which a different entity is forced to proceed at its maximal speed whenever possible. The state space thus explored is not exhaustive. Nevertheless, protocol design errors such as deadlocks, unspecified receptions, and channel overflows can still be detected. In addition, this method has another advantage over others in that it can be structured to run as two processes on two processors to further speed up the validation process.

8. The relief strategy proposed by Itoh and Ichikawa (1983) is applicable to protocols whose entity FSMs do not contain any cycle not passing the initial states. In each global state, the admissible events of different entities are executed simultaneously to derive the next global state. Moreover, if there is some potentially admissible event in the current state of an entity E, additional global state derivations by inhibiting the execution of all the admissible events of E should also be performed. The purpose is to force entity E to wait in order that any of its potentially admissible events may become executable later. Following this algorithm, only part of the global state graph is explored. The interaction sequences thus explored are called the *reduced implementation sequences* and are used to verify the protocol against the given requirement specification.

9. This group of strategies (Brand and Zafiropulo, 1983; Kakuda *et al.*, 1986) are called the *tree (or acyclic form) protocol validation.* Instead of exploring the global states of a protocol, this strategy grows each entity of the protocol into a tree or an acyclic form. During the growing process, protocol design errors such as unspecified receptions, deadlocks, and channel overflows can be detected. The algorithm of this strategy is much more complicated than the traditional "global states" reachability analysis. Nevertheless, the validation speed is improved.

10. Holzmann (1985, 1987) designed a tool called *Trace*, which also works under two modes, either as a fast debugging tool or as a slower correctness prover. The main emphasis is that the user can control the scope of each search. When used as a debugging tool, Trace uses a search strategy called *scatter search* to explore the global states graph, which basically is a depth-first search guided by some simple heuristics and restricted by a depth limit. Examples of the heuristics proposed by Holzmann are as follows:

(a) Restrict the amount of nondeterminism.
(b) Assign priorities among concurrent events.
(c) Limit queue sizes.
(d) Discard all the states after the depth-first exploration except those that are *loop states.*
(e) Keep a limited size of cache for storing global states.
(f) Minimize the FSM models of protocol entities before verification.

11. This strategy (West, 1986) is called the *random-walk state exploration.* From his experience in validating the OSI session layer protocol, West observed that the majority of errors detected are found many times in different global states for a complex protocol. This sugests that an analysis of a subset of the reachable global states may be sufficient to identify a significant fraction of errors. The random-walk strategy is thus proposed as a way to partially

explore the global states graph. The strategy is as follows:

(a) If there is any event that may cause message collision when executed, such an event is fired first; otherwise, arbitrarily choose any event to fire.
(b) The state exploration is stretched out continuously along a single path without backtracking. As a result, none of the previous states needs to be remembered and the states that have already been explored may be explored again.

12. Vuong *et al.* (1986) proposed a new global state representation based on which the reachability algorithm developed can generate "finite graphs" for all non-FIFO and for a certain class of FIFO protocols even though these protocols may produce an unbounded number of messages in the transmission media. This approach thus solves a class of problems that the conventional reachability analysis fails to deal with due to the infinity of the reachable global states induced by unbounded accumulation of messages in the media.

4.3 PROVAT Strategy

Most of the relief strategies described in Section 4.2 are ad hoc, utilizing heuristic information. We believe that the problem should be attacked more systematically by borrowing some ideas from the search strategies developed in the field of Artificial Intelligence (AI). Instead of adopting any of the aforementioned strategies in our validation tool, we have developed our own from a new approach. We call the strategy PROVAT (PROtocol VAlidation Testing) for the following two reasons (Lin *et al.*, 1987):

1. It is a strategy incorporated into a validation tool.
2. When the tool resorts to the PROVAT strategy, it is performing a task of *design testing* instead of *design validation* since only some of the reachable global states will be explored. The purpose is to show the existence, *not* the absence of protocol design errors.

Compared to general search problems, the search done on the state space of a protocol has the following distinguished features:

1. Rather than searching for an optimum or satisfactory solution, Validation Testing searches for protocol design errors of unspecified receptions, deadlock states, and channel overflows.
2. The quality of search strategy is judged by the discovered percentage of the total number of errors in a limited amount of time and space. Better

strategies will discover higher percentages of errors in the same amount of time and space.

3. When searching into the protocol state space, pruning can be done based on how likely a subtree of states can be exercised by the protocol operation. In case an exhaustive analysis is infeasible, those states that are more frequently exercised by the protocol should be validated first.
4. An effective search will primarily focus on one type of error at a time because the heuristics required in locating different types of errors may contradict each other.

Like the *best first search* developed in the AI field, an ideal search in the domain of protocol validation is called the *error first search.* PROVAT is a first attempt to characterize such an error first search.

As pointed out in the previous section, heuristics can be applied at three points in a search process, namely, the points to decide which global states to expand next, to decide which transitions to fire next, and to decide which global states to discard. PROVAT is designed to employ heuristics at all three points.

Following the definitions given by Peral (1984), a global state is said to be *generated* when its data representation is computed from that of its parent. When this occurs, the parent state is said to be *expanded.* A state is *fully expanded* if all of its children are generated; otherwise, it is *partially expanded.* At some point, each generated state has to be inspected to see whether it reveals any of the protocol design errors. A state is called *explored* if it has been inspected. In addition, during the validation process, the states generated are dynamically partitioned into two sets: CLOSED and OPEN. In the search algorithm of PROVAT, the generated states that are never or partially expanded are placed in OPEN, and those that have been fully expanded are moved to CLOSED. Since a state may remain in OPEN for a long time before it is expanded, it is reasonable to explore the state immediately after it is generated.

In the following, the heuristics used by PROVAT are explained. We assume that the only available protocol operations are "send" and "receive."

1. *Heuristics in Deciding Which Global States to Expand Next.* The purpose is to expand those global states in OPEN that are closer to errors. The heuristics are mainly concerned with the status of queues and entities. For each type of protocol errors, a different heuristic is derived.

(a) *Unspecified Reception.* We count the number of queues that satisfy the following two conditions: (1) the queue is nonempty, and (2) its destination entity is willing to receive the message at the head of the

queue. Global states having the largest number of this type of queue will be expanded first.

(b) *Deadlock.* Examine all the empty queues in a global state, and call N_1 the number of empty queues whose destination entities are in receiving states (states without any outgoing "send" transitions), and N_2 the number of empty queues whose destination entities are not. Global states then are scored according to the weighted sum of N_1 and N_2. The states that receive the highest score will get the first attention.

(c) *Channel Overflow.* Global states are compared based on the length of their longest queue. If a tie occurs, the comparison continues based on the length of the second longest queue. States that win in this contest will be explored first.

2. *Heuristics in Deciding Which Transitions to Fire Next.* The purpose is to perform those actions that are more likely to lead to the error from a selected state. The heuristics are concerned with either action types or queue lengths. Different heuristics are developed for each type of error.

(a) *Unspecified Reception.* We choose a "receive" operation if that operation is able to receive a message from the shortest queue that contains at least two messages; otherwise, we choose a "send" operation that sends a message to an empty queue. For other operations, consider "receive" before "send." These heuristics tend to sustain the decision made by the heuristics of choosing the next expanded global state.

(b) *Deadlock.* "Receive" operations are always considered first. Among "receive" operations, we choose those which extract from the shortest queue.

(c) *Channel Overflow.* "Send" operations are always considered first. Among "sends," those which add to the longest queue have the highest priorities. If there is no send operation, "receives" that extract messages from the shortest queue are chosen. The heuristics for the above two types of error are also derived to be compatible with those in deciding which global states to expand next.

3. *Heuristics in Deciding Which Global States to Discard.* The purpose is to decide which global states should not be generated during the global state expansion, which in fact prunes the subtree rooted by any state thus inhibited. To bring in meaningful heuristics in making this decision, we first enhance the original TG model (Lu, 1986), on which the validation tool is based, to include probability specifications. Then a simple method is developed to estimate how likely each of the global states will be reached in terms of probabilities (similar work is done on CCS by Purushothaman and Subrahmanyam (1987) for a

different purpose). Global states being assigned smaller probabilities are less likely to be reached by the protocol operation. Consequently, if speed is the major concern to the protocol verifier, he or she can ask PROVAT to explore only those states with probabilities of reachability higher than a specified threshold.

All the heuristics informally defined above are quantified by the *evaluation functions*, which play major roles in guiding the reachability analysis.

Another problem left out in the above discussion is when to terminate the partial state exploration. In PROVAT, this is decided by two criteria:

1. *Specifying a probability threshold to explore only the states that are more likely to be exercised by the protocol.* Those global states with probabilities of reachability dropped below the threshold value will never be generated.
2. *Specifying an upper bound on the number of expansion steps.* When a state is expanded, some new or existing states will be generated. Each step of generating a state, whether the resulting state is new or already existing, is called an expansion step. When the number of expansion steps exceeds the specified value, the analysis terminates.

The first criterion essentially is supported by the third kind of heuristics discussed above. On the other hand, the second criterion gives an approximate estimate on how much time will be taken by the analysis. In PROVAT, the first criterion is used to tailor the state space to contain only those paths that are more likely to be exercised by the protocol, then the second criterion is applied to obtain a desirable response time.

4.4 Preliminary Results

The PROVAT strategy has been built into an exhaustive validation tool based on the formal model of transmission grammar (TG) (Teng, 1980; Umbaugh, 1983; Lu, 1986). In order to incorporate PROVAT into the TG validation tool, the model is first enhanced to include probability specifications and is called PTG (Probabilistic Transmission Grammar) (Lin, 1988). Then a part of the tool is recoded to encompass the PROVAT strategy based on the new model.

The original TG validation tool runs under 4.3BSD UNIX and contains about 3800 lines of C language code. The resulting PTG tool contains about 4500 lines of C code. Several real-life protocols have been extensively validated

and tested to evaluate the effectiveness of PROVAT. Here only the tests performed on the call establishment procedure of the CCITT X.21 as specified in (West and Zafiropulo, 1978) are presented. Though the X.21 interface is designed to operate in a physical environment where no more than one message can be outstanding in the channel of either direction, as an exercise to compare PROVAT with the other search strategies in a large state space, the behaviors of the X.21 with other channel sizes are also tried. Also, when testing the power of the heuristics for deadlock detection, we delete some of the transitions from the X.21 specification in (West and Zafiropulo, 1978) to create deadlock states uniformly scattered in the global state space.

Five search strategies are tested to compare their performance in locating different types of design errors:

1. D-search (DS for short).
2. Depth-first search (DFS for short).
3. Heuristic search based on state heuristics only (abbreviated SBHS: State-Based Heuristic Search).
4. Heuristic search based on transition heuristics only (abbreviated TBHS: Transition-Based Heuristic Search).
5. Heuristic search based on both state and transition heuristics.

Notice that PROVAT adopts the fifth search strategy, whereas the TG validation tool used the first strategy. Also, the second, third, and fourth strategies are special cases of the fifth.

The results of the X.21 testing are shown in Figs. 14 to 16 and Tables I to IV. In these results, state pruning is not considered though it is also part of PROVAT. Figure 14 compares the results of detecting unspecified receptions for the aforementioned five search strategies when the channel size of the X.21 is set to be 1, which clearly shows the power of heuristics in guiding the "error first" search. Then, in Fig. 15, the results of the PROVAT search and the D-search in detecting unspecified receptions for the X.21 with channel size of 3 are compared, which exhibits the improvement of PROVAT over D-search. Finally, Fig. 16 shows the superiority of the PROVAT search over all the other strategies in quickly locating the first 20% of reception errors when the channel size is also 3.

Tables I to IV compare the results of detecting deadlocks for the five search strategies under the assumption of different channel sizes. Each table entry gives the expansion steps needed to discover one more deadlock error. It is interesting to note that when the channel size is 4, PROVAT exercised only 3.4% of total expansion steps needed for an exhaustive search in order to locate all the deadlock states. These results give a strong evidence that with

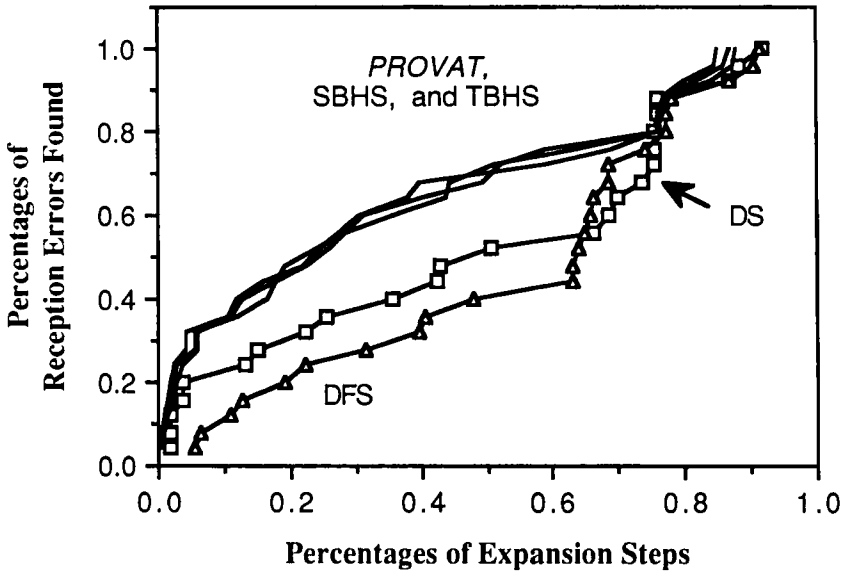


FIG. 14. Detection of unspecified reception errors with channel size 1

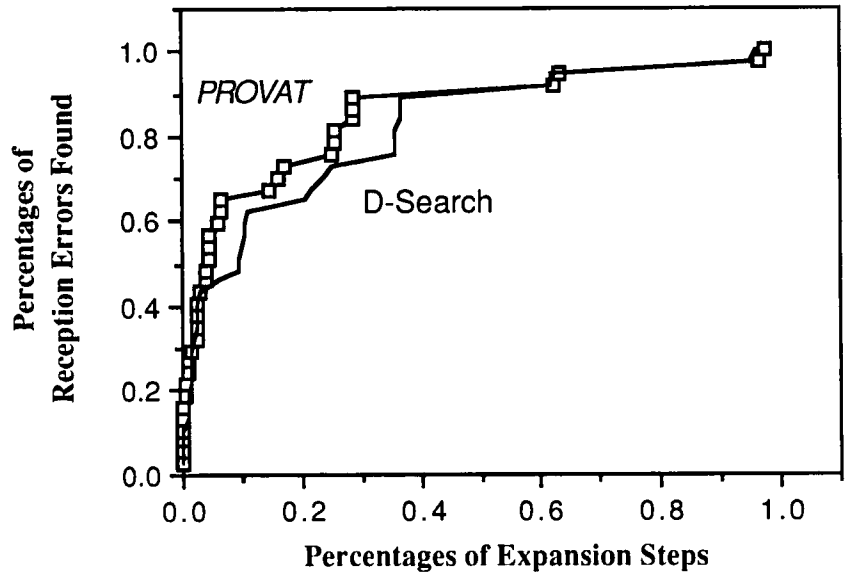


FIG. 15. Comparison of PROVAT vs. D-search in detecting reception errors

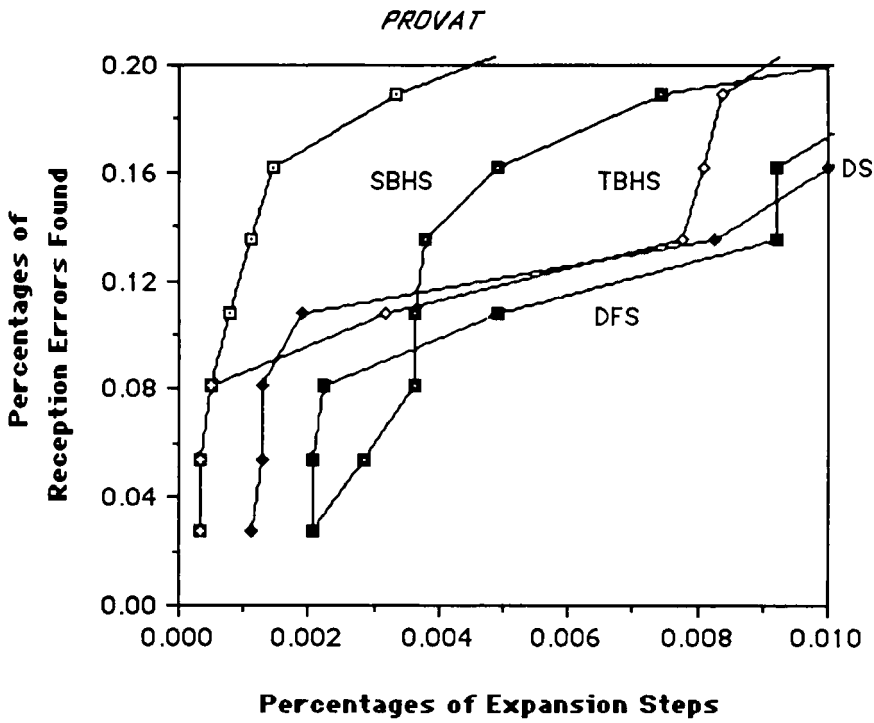


FIG. 16. Detection of the first 20% of unspecified reception errors

PROVAT incorporated, the validation tool is much more effective than blindly performing the D-search used by Lu (1986).

The experimental results from these tests are optimistic in that when we resort to PROVAT, it does help locate the errors in fewer steps than the other strategies. Though these results are still insufficient to conclude that PROVAT will also perform better in validating other protocols, it indicates that with good heuristics the verification tool may do a better job when state explosion prohibits a thorough analysis of protocol behavior.

The heuristics employed by PROVAT are based on the local information of a global state. Thus only a little overhead is incurred in the reachability algorithm. Though it seems difficult, if not impossible, to capture the characteristics of the interactions leading to a protocol design error by some heuristics, PROVAT has shown its effectiveness through our experimentation.

It is also worth noting that among the heuristics of locating unspecified receptions, deadlock states, and channel overflows, those for unspecified receptions are the most difficult to capture. This seems to match with the

TABLE I

THE DETECTION OF DEADLOCK STATE ERRORS WITH A TOTAL SPACE OF 280 STATES

Search Strategy / Deadlock Error	Channel Size = 1				
	DS	DFS	TBHS	SBHS	PROVAT
1	20	18	16	6	4
2	23	118	118	17	16
3	212	119	119	159	155
4	382	301	299	259	259
Percentages of Steps Excercised	94.3%	74.3%	73.8%	64%	64%

TABLE II

THE DETECTION OF DEADLOCK STATE ERRORS WITH A TOTAL SPACE OF 946 STATES

Search Strategy / Deadlock Error	Channel Size = 2				
	DS	DFS	TBHS	SBHS	PROVAT
1	20	18	16	6	4
2	23	221	221	19	16
3	336	222	222	220	212
4	1408	767	764	394	389
Percentages of Steps Excercised	98%	53.4%	53.2%	27.4%	27%

TABLE III

THE DETECTION OF DEADLOCK STATE ERRORS WITH A TOTAL SPACE OF 3237 STATES

Search Strategy / Deadlock Error	Channel Size = 3				
	DS	DFS	TBHS	SBHS	PROVAT
1	20	18	16	6	4
2	23	271	271	19	16
3	4930	272	272	260	248
4	5178	1374	1370	513	508
Percentages of Steps Excercised	99.6%	26.4%	26.4%	9.8%	9.8%

TABLE IV

THE DETECTION OF DEADLOCK STATE ERRORS WITH A TOTAL SPACE OF 11054 STATES

Search Strategy / Deadlock Error	Channel Size = 4				
	DS	DFS	TBHS	SBHS	PROVAT
1	20	18	16	6	4
2	23	290	290	19	16
3	17953	291	291	280	268
4	18275	2027	2023	624	619
Percentages of Steps Excercised	99.8%	11%	11.1%	3.4%	3.4%

research efforts in protocol synthesis (see Section 6) where incompleteness of receptions is always the major issue (Zafiropulo *et al.*, 1980).

Though it is difficult to compare the effectiveness of PROVAT strategy with that of the other approaches due to the lack of a common ground, the advantage of PROVAT lies in its simplicity and systematic approach. Its drawback is common to any heuristics-based approach in its lack of theoretic support and predictability. Our experience shows that to perform protocol validation via reachability analysis, care must be taken from the beginning. As the classification we made for the relief strategies indicates, serious attention should be paid to the early stages of validation such as modeling and function abstraction/decomposition. Only through these combined efforts, the difficult state explosion problem can be resolved more effectively.

5. Verification and Conformity Analysis

As mentioned in the previous section, protocol validation and verification is a demonstration of the correctness of a protocol design. A protocol is considered to be correct if it satisfies two kinds of properties, viz., syntactic properties (or general properties) and functional properties (or specific properties). The syntactic properties are those desired properties common to all protocols such as freedom from deadlock, completeness and progress. They form the set of implicit requirements that any protocol should fulfill to ensure that its logical structure has no syntactical errors. The absence of syntactical errors, however, does not necessarily imply that the protocol will do what it is supposed to do. In this regard, the functional properties of a protocol define the specific objectives of the protocol. They are usually presented in terms of a set of behaviors, called the *communication service*, as perceived by the protocol users. As mentioned earlier, a protocol can engage in extremely complicated interactions that are beyond human anticipation. A formal analysis is required to ensure that the functional behavior of a protocol conforms to the designer's intention.

To date, while the syntactic properties of protocols have been extensively studied and relatively well understood (see Section 4), much work remains to be done on the functional analysis, also called conformity analysis. In this section, we will briefly survey the work done in this area and present our approach to it.

5.1 Service Concept

The service concept is receiving more and more attention in current protocol design (Vissers and Logrippo, 1985). With the abstraction facility in service specification, the complexity problem of protocol design can be

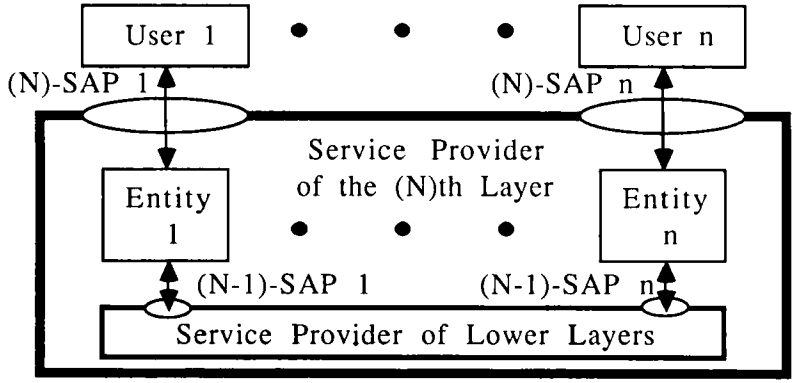


FIG. 17. Architectural model for layered protocol design

alleviated to such an extent that protocol designers are capable of dealing with it competently. Employing the service concept, the architectural model for layered protocol design in the OSI Reference Model (see Fig. 2) is elegant and succinct. The architecture model shown in Fig. 17 may be considered as an abstraction of the network architecture shown in Fig. 2.

As explained in Section 2.3, services represent the logical interfaces between adjacent layers, while protocols represent the operations performed inside layers. Accordingly, the service specification and the protocol specification describe the behavior of a system at two different levels of abstraction. A service specification is responsible for defining the valid sequences of interactions visible at the boundary between two adjacent layers, whereas a protocol specification defines the behavior of protocol entities inside a particular layer in terms of the interactions between peer entities.

Referring to Fig. 17 for a specific protocol layer, say the Nth layer, the communicating entities together provide a set of capabilities to the service users through the (N)-Service Access Points (or (N)-SAPs for short) by obeying the (N)-protocol and by making use of the service provided by the layers below this one. In other words, the (N)-protocol combined with the service provided by lower layers forms a service provider to the service users, which may be end users or the communicating entities in the next higher layer, ie., the $(N + 1)$th layer. Consequently, the (N)-protocol can be regarded as the logical implementation of the (N)-service given the $(N - 1)$-service available for use. Since the (N)-SAPs are the only places through which the (N)-service can be accessed by the service users, the internal mechanism embedded in the protocol and the interaction between communicating entities are not visible to the service users. For example, the Alternating Bit Protocol (ABP) provides a service that guarantees the correct transfer of data in sequence from one

user to the other. However, the use of an alternating bit variable in each communicating entity and the retransmission mechanism in the communicating entity serving the user that has data for transmission are not visible to both service users.

The set of capabilities provided by the communicating entities in a protocol layer is presented by the execution of a group of well-defined service primitives. A service primitive is considered as an elementary interaction between a service user and the service provider during which certain values for the various parameters of the primitive are established to which both the user and the provider are refer. Thus each (N)-service primitive is associated with an (N)-SAP and executed at that (N)-SAP. The specification of an (N)-service can be expressed in terms of the possible orderings of service primitives associated with the (N)-SAPs and their parameter value dependencies (Vissers and Logrippo, 1985). On the other hand, the specification of the (N)-protocol can be expressed in terms of the possible orderings of service primitives associated with the (N)-SAPs and the ($N - 1$)-SAPs and their parameter value dependencies.

Several advantages of the service concept are as follows:

1. The main advantage of utilizing the service concept in communication protocol design is to provide a framework on which the complexity of protocol design can be better managed.
2. A protocol designed using the service concept can be changed without affecting any layer other than the one the protocol resides in. This is due to the principle of separation of concerns in the service concept.
3. Yet another advantage of using the service concept in protocol design is to facilitate the correctness proofs. Without the service concept, the verification of a communication system becomes an unsurmountably difficult task.

5.2 Conformity Analysis

By *conformity analysis* it is meant to demonstrate that a protocol does indeed provide the service for which it is intended. The purpose of the conformity analysis is to show that the composite behavior of the (N)-protocol specification and the ($N - 1$)-service specification with respect to the upper users conforms to the (N)-service specification. Consequently, any method for conformity analysis should be able to establish properties of the communication behavior of a given specification, to integrate several specifications into an overall behavior, to hide the internal communications, and to demonstrate the equivalence of two communication behaviors.

As mentioned earlier, the service specifications and protocol specifications represent two levels of abstraction in the OSI Reference Model. At the higher level, a service specification of a layer describes the externally visible service in terms of the valid sequences of the interactions taking place at the upper boundary. At the lower level, a protocol specification describes the logical implementation of a service in terms of the behavior of the protocol entities inside a layer. Due to their inherently distinct characteristics, past experience has shown that *sequence-oriented* specification techniques are more suitable for service specifications, whereas *state-oriented* specification techniques are better for protocol specifications. A good survey on a spectrum of various specification methods is in (Schwartz and Melliar-Smith, 1982).

Although different description methods have been commonly used for service specifications and protocol specifications, a single specification technique to describe both of them is needed to perform conformity analysis. First, by using one technique, both the service and protocol specifications can be interpreted and analyzed on a common semantic basis. Second, a major task in the conformity analysis involves the composition of the protocol specification at one layer and the service specification at the lower layer. A uniform specification technique will facilitate this composition step.

We have developed a CSP-based language for both the service and protocol specifications (Liu and Liu, 1984). The basic idea is summarized as follows:

1. *To specify a service*, one or more CSP processes are used to describe the behavior of a service provider. Furthermore, these processes can only communicate with the processes that represent the service users. In this way, a CSP specifications can be viewed as a *communication sequences generator* in the sense that a service is defined in terms of all the possible communication sequences that may arise during its execution.
2. *To specify a protocol*, the entities are described by CSP processes that may communicate with the processes that represent the underlying service provider and the upper users. Typically, a control point in a process just before an input command reflects the major (or control) state of a protocol entity, whereas the variables are used to represent the "context variables" associated with a protocol entity, such as the messages. Therefore, a CSP specification can be viewed as a *state-transition machine.*

In short, our experience has shown that, by using CSP in a disciplined manner, one can make use of language constructs as mechanisms in generating a set of communication sequences corresponding to the allowable sequences of interactions of a service. On the other hand, one can also specify a protocol entity as a CSP process that resembles a state-transition machine

with the state space determined by the variables and a set of control points in the process. Therefore, even though CSP is a high-level language, we can employ it as a unified method of sequence-oriented techniques and state-oriented techniques that can be used for both service and protocol specifications.

In the context of CSP, if the (N)-entities and $(N-1)$-service provider are specified as a set of processes, the task of conformity analysis is to show that, after hiding all the internal communications, the set of observable communication sequences of these processes with respect to the users should *conform* to the set of communication sequences exhibited by the processes of the (N)-service provider. There are two approaches to conformity analysis based on CSP specifications. We will discuss each of the approaches in the following subsections.

5.3 Axiomatic Approach

For CSP, a number of proof systems have been proposed (Apt *et al.*, 1980; Levin and Gries, 1981; Soundararajan 1984). While each of these provides a different way to prove correctness of the distributed programs written in CSP, all are based on Hoare's axiomatic approach (Hoare, 1969). In this approach, one can make use of a set of axioms and inference rules to prove that the behavior of a program has some desired properties.

The axiomatic approach has been considered a successful tool in the design of sequential programs. Given an initial condition that is satisfied at the beginning of a program, the prover can systematically derive the logic assertions at different control points, ultimately establishing a desired postcondition at the end of the program. The application of the axiomatic approach to distributed programs is, however, far from well understood. Unlike the simple input/output behavior presented by a sequential program, a distributed program usually has a number of interacting processes that are mutually dependent in the course of their executions. Despite the many techniques that have been proposed to tackle the new problems associated with distributed programs, more experience is needed before they can be of practical use.

Besides the fact that the axiomatic approach is still in the experimental stage, there are certain fundamental difficulties that prevent us from using this approach for the conformity analysis of communication protocols, as described below:

1. Most of the axiomatic-based systems can deal only with partial correctness of CSP programs. In other words, they are used to prove that

certain properties will hold after the execution of a program, provided that it terminates. In contrast, we are interested in the communication sequence patterns presented by systems that usually involve *infinite computations.*

2. In axiomatic-based systems, a program behavior is described by a set of logic formulas. In general, it is difficult to guarantee that these formulas can completely characterize the properties of the program; they can at best serve as a substantial but incomplete description of the program behavior. However, to establish the equivalence of two program behaviors, the *strongest* descriptions of the programs are required.
3. The axiomatic-based systems aim at proving some desired properties of a *CSP program,* i.e., a closed set of processes communicating with each other. However, we are concerned with the external behavior of a set of processes that may interact with the environment, i.e., an *open system.*

Rather than taking the axiomatic approach, we have developed a transformational approach for the conformity analysis of communication systems, which is given in the following subsection.

5.4 Transformational Approach

The basic idea of our approach is as follows (Liu and Liu, 1986). Instead of performing the logic reasoning on the CSP processes, we transform a CSP process into a set of algebraic expressions. These algebraic expressions should represent the complete description of the communication behavior of the original process. Furthermore, the algebraic system itself should be equipped with the appropriate operators to support the activities of conformity analysis. Once we achieve this, we are able to perform the analysis of a set of CSP processes by simple *algebraic manipulations* of their derived expressions.

The immediate advantage of this approach is that, in general, algebraic manipulations can be carried out more systematically and mechanically than the mathematical logic inferences performed in the axiomatic approach. However, in order to obtain this advantage, a major premise is that the transformation from CSP processes to algebraic expressions should be performed in a simple and orderly manner. For this purpose, we developed a transformation system consisting of a set of rules by which the transformation is conducted. Milner's Calculus of Communicating Systems (Milner, 1980) was chosen as the target language of our transformation system for the following reasons:

1. CCS bears many similarities with CSP, thus making the transformation system simple and straightforward. In particular, the concept of

"interaction" in both languages is based on synchronous communication.

2. Besides being an elegant notation for describing communication behaviors, CCS provides a set of operators to manipulate communication behaviors. Especially, the composition operator can be used to derive the integrated behavior of a set of cooperating system components, while the restriction operator can be used to hide internal communications.
3. CCS is associated with a sound underlying theory to show the equivalence of two communication behaviors—an essential activity in conformity analysis.

To perform the conformity analysis, we have developed a transformation system to extract from a CSP process the communication sequences that may arise during its execution, and to express these sequences in terms of behavior expressions in CCS. Based on this system, we are able to transform a set of cooperating CSP processes into a set of CCS expressions, and then derive the integrated behavior with respect to the environment by using the CCS composition and restriction operators.

Also, the conformity of the (N)-protocol to its service can be shown by proving that the CCS expression, representing the integrated behavior of the (N)-entities and $(N-1)$-service provider with respect to the users at the next layer, is observation-equivalent to the CCS expression that represents the behavior of (N)-service provider.

The overall steps in conformity analysis are outlined in Fig. 18. We have used the transformation system to verify the functional properties of the ABP. In addition, it was used to detect syntactic errors of the X.25 packet-level DCE/DTE interface. The details can be found in (Liu, 1986).

The transformation system from CSP to CCS is quite straightforward and syntax-directed. For a given CSP process, the system allows us to suppress the local computations and deal with its communication behavior only. In particular, for a process that performs cyclic operations, the derived CCS expression can serve as an "invariant" property on its communication behavior. This gives some advantage over axiomatic proof systems, since by using these systems, it is the prover who has the responsibility to elaborate the invariant properties for cyclic computations—a heavy burden.

In passing we like to point out that the transformational approach (Partsch and Steinbruggen, 1983) has been used for software development. That is, starting with a formal specification, a transformation process is performed for transforming the specification into an implementation. In contrast, we use the transformational approach to derive from the specifications the properties of communication behaviors in terms of algebraic expressions, which are subsequently used for functional analysis.

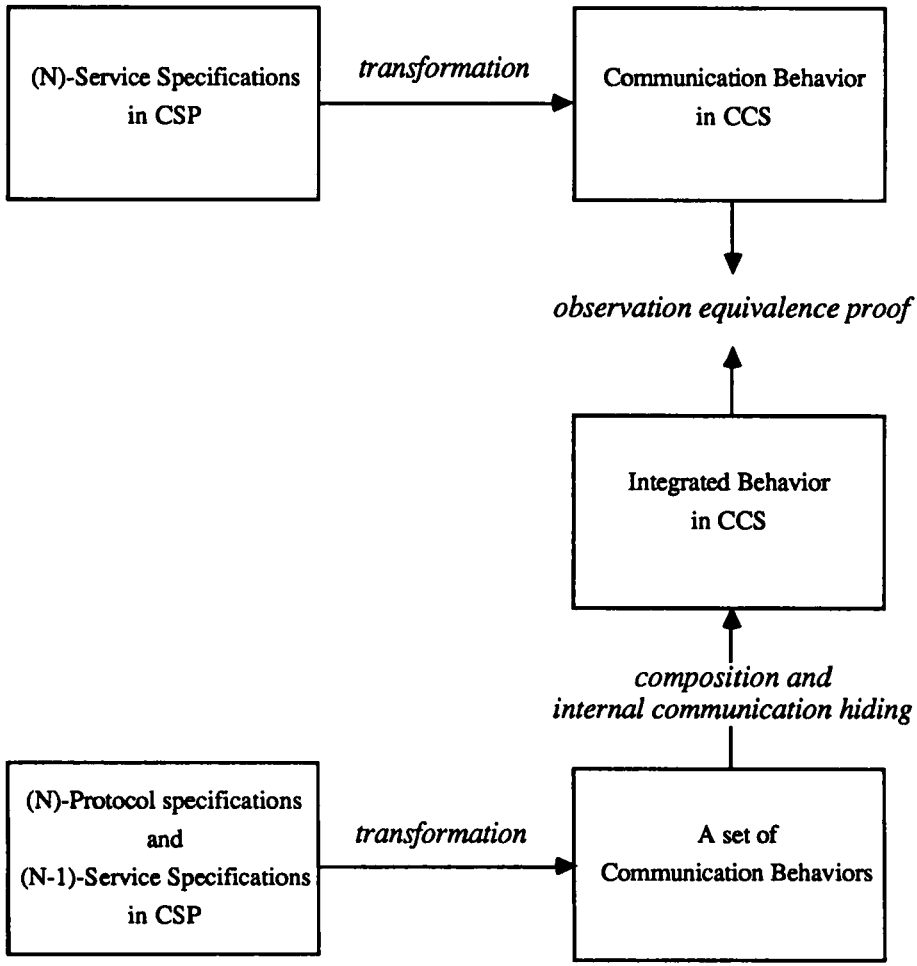


FIG. 18. An overview of conformity analysis

6. Protocol Synthesis

As mentioned in Section 1, two complementary approaches to ensuring correctness of computer-communication protocols are analysis and synthesis. By the analysis approach, a protocol is first examined to reveal some properties, desirable or undesirable, and then modified to get rid of the undesirable ones (see Sections 4 and 5). By the synthesis approach, rules ensuring some desirable properties are enforced during the protocol design process.

The synthesis approach has the advantage over the analysis approach in that it can assist the protocol designer to reduce the possibility of making errors, if not to prevent it totally, during the protocol design process. In this section, we will briefly survey the work done previously by researchers in the area of protocol synthesis, discuss the limitations of current protocol synthesis techniques, present our protocol synthesis technique, and discuss future work on this topic.

6.1 Previous Work

Previous work on protocol synthesis can be classified into two categories, depending on whether a service specification (see Sections 2, 3 and 5.1) is required or not.

6.1.1 No Service Specification Required

Protocol synthesis techniques in this category do not require the initial existence of a service specification to which the synthesized protocol specification has to conform. Therefore, the protocol designer is responsible for the semantics of the synthesized protocol specification. The goal of these techniques is to construct protocol specifications free from the following *logical errors*: nonspecified reception, nonexecutable interaction, deadlock, unboundedness, and improper termination. Each technique has achieved either a portion or the whole of the goal. Generally speaking, the techniques achieving just a portion of the goal have higher flexibility than those achieving the whole of the goal. Seven techniques are included in this category, each of which is discussed in the following:

1. *Zafiropulo's Reception Production Rules.* Zafiropulo *et al.* (1980) proposed three reception production rules, which were used in an interactive protocol synthesis system (see Section 10.1). As long as these rules are obeyed, two protocol logical errors—unspecified reception and nonexecutable interaction—can be prevented for any synthesized protocol specification. These rules, however, are only applicable to two-entity protocols. To handle multi-entity protocols, Brand and Zafiropulo (1980) proposed a different set of production rules which are much more complicated than those for two-entity protocols. Protocol logical errors such as deadlock, though not preventable, may be monitored by the system in the process of designing a protocol. The internal representation of protocol behavior in the system is N trees for an N-entity protocol.

2. *Sidhu's Protocol Design Rules.* Sidhu (1982a) proposed four protocol design rules that can be used to monitor all kinds of protocol logical errors.

However, the protocol designer has to specify all the interactions (message transmissions and receptions) between communicating entities. Thus the technique is just an algorithm to validate a protocol in the process of designing it and is not a real synthesis technique. The internal representation of protocol behavior in the technique is a global state-transition graph.

3. *Zhang's Protocol Synthesis Algorithm.* The protocol synthesis algorithm proposed by Zhang *et al.* (1988a, 1988b) consists of three production rules and two deadlock avoidance rules. Like Sidhu's protocol design rules, the internal representation of protocol behavior in his algorithm is a global state-transition graph. Their technique can be considered as an improvement over Sidhu's technique in that they enhanced Sidhu's technique by automatically generating the specifications of all receptions that can occur and by adding deadlock avoidance rules to prevent possible occurrence of deadlock. Their technique is restricted to two-entity protocols and it is suspected that the deadlock avoidance rules are not general enough to cover all deadlock-free two-entity protocols.

4. *Choi's Sequence Method.* Choi (1986) presented a method for constructing protocol specifications in the Finite State Machine (FSM) model by first synthesizing a pair of regular expressions of star height zero or one and then converting the regular expressions to equivalent FSMs. His method can prevent all kinds of protocol logical errors mentioned above. However, his technique is limited to two-entity protocols whose entity FSMs correspond to regular expressions of star height at most one.

5. *Gouda's Synthesis Algorithm.* Given a partial specification of a communicating entity, the algorithm proposed by Gouda and Yu (1984b) enforces a fixed communication pattern between two communicating entities in order to construct the complete protocol specification in which all kinds of design errors are not existent. One disadvantage of their algorithm, is that the generated specification for the peer entity is just one of the possible correct specifications and may not be the one intended by the protocol designer. Furthermore, the algorithm is applicable only to two-entity protocols.

6. *Ramamoorthy's Automated Protocol Synthesizer.* The automated protocol synthesizer developed by Ramamoorthy and his associates (Ramamoorthy and Dong, 1982; Ramamoorthy *et al.*, 1985) makes use of six transformation rules to build up the specification for the peer entity from a given specification for the local entity. All kinds of design errors can be prevented by this synthesizer if the specification for the local entity possesses some desirable properties. The synthesizer suffers the same drawbacks as Gouda's algorithm. We will discuss this system in more detail in Section 10.3.

7. *Kakuda's Component-Based Synthesis.* Kakuda and Wakahara (1987) generalized Ramamoorthy's six rules to come up with 22 patterns of components, which may be used to construct multi-entity protocols. Moreover, this technique allows the protocol designer to interactively increase flexibility for protocol construction. All kinds of protocol logical errors can be prevented.

6.1.2 Service Specification Required

Protocol synthesis techniques in this category require the initial provision of a service specification to which the synthesized protocol specification has to conform. The goal of these techniques is not only to construct protocols free from protocol logical errors, but also to mandate the synthesized protocol specification to conform to the given service specification (see Section 5.2). In the following, we briefly describe three such techniques.

1. *Merlin's Submodule Construction Method.* Merlin and Bochmann (1983) proposed a method of determining the specification for the missing entity from a given service specification and the specifications for the remaining entities. Unfortunately, the technique does not guarantee the deadlock-freedom for the synthesized protocol specification and thus must be supplemented by an analysis procedure to detect the deadlock.

2. *Prinoth's Protocol Construction Algorithm.* The input to Prinoth's protocol construction algorithm (Prinoth, 1982) is actually a specification refined from a service specification by adding some auxiliary action transitions, and the output from the algorithm is a protocol specification. Therefore, the protocol designer has to refine the service specification to produce the input to the algorithm. The algorithm itself does not include a method to perform the refinement of the service specification.

3. *Bochmann's Protocol Derivation Algorithm.* Bochmann and Gotzhein (1986) proposed an algorithm to derive a protocol specification from a given service specification. A service in his model is described by an expression of service primitives connected by sequence, parallelism, and alternative operators. A syntax tree is employed to collect the necessary information for the send and receive actions required for synchronizing service primitives. Consequently, their specification language is not able to describe a service containing an infinite number of possible execution paths. Inclusion of a recursion operator, as suggested in their paper, may fill the deficiency but may also complicate their algorithm to some extent.

6.1.3 Comparison and Discussion

The protocol synthesis techniques in the first category (Section 6.1.1) provide some rules or methods for obtaining the complete protocol specification, starting from a partial protocol specification, either interactively or fully automatically. However, they don't have a service specification initially given as a reference. The protocol designer is responsible for the semantics of the synthesized protocol specification; thus he or she must resort to his or her intuitive understanding of the intended service, a very informal task in current protocol design. As a result, more burden is placed on the protocol designer in the stage of protocol verification.

The protocol synthesis techniques in the second category (Section 6.1.2) do consider service specifications in a formal manner. Merlin's work, however, additionally requires the existence of specifications for $(n-1)$ communicating entities, where n is the total number of communicating entities in the protocol layer of interest. Prinoth's work and Bochmann's work are more ambitious since only the service specification of the interest layer is needed at the outset. Nevertheless, in Prinoth's work, some auxiliary actions (similar to the synchronization messages discussed in Section 6.2) are, in some cases, needed to be added into the service specification prior to the application of his protocol construction algorithm; yet the algorithm does not provide a method to perform the refinement of the service specification by including such auxiliary actions. In Bochmann's work, the required synchronization messages are derived automatically; however, their service specification language is not able to express a service containing an infinite number of possible execution paths. In our protocol derivation algorithm given below, we follow the same approach of Bochmann's work, and thus inherit the advantages of his approach. But we use a state-transition model, which can easily describe a service containing an infinite number of possible execution paths by using transition loops in FSMs and which seems to be a more natural and better understood model. In the next subsection, we will briefly explain our protocol synthesis technique.

6.2 Our Synthesis Technique

We believe the right approach to protocol design should be one that treats the service concept formally. In particular, we feel that one should start from a formal specifications of the (N)-service and the $(N-1)$-service to construct the desired formal specification of the (N)-protocol, as depicted in Fig. 19. Within this architectural view, we are interested in automating the procedure of deriving a protocol specification from given service specifications. That is,

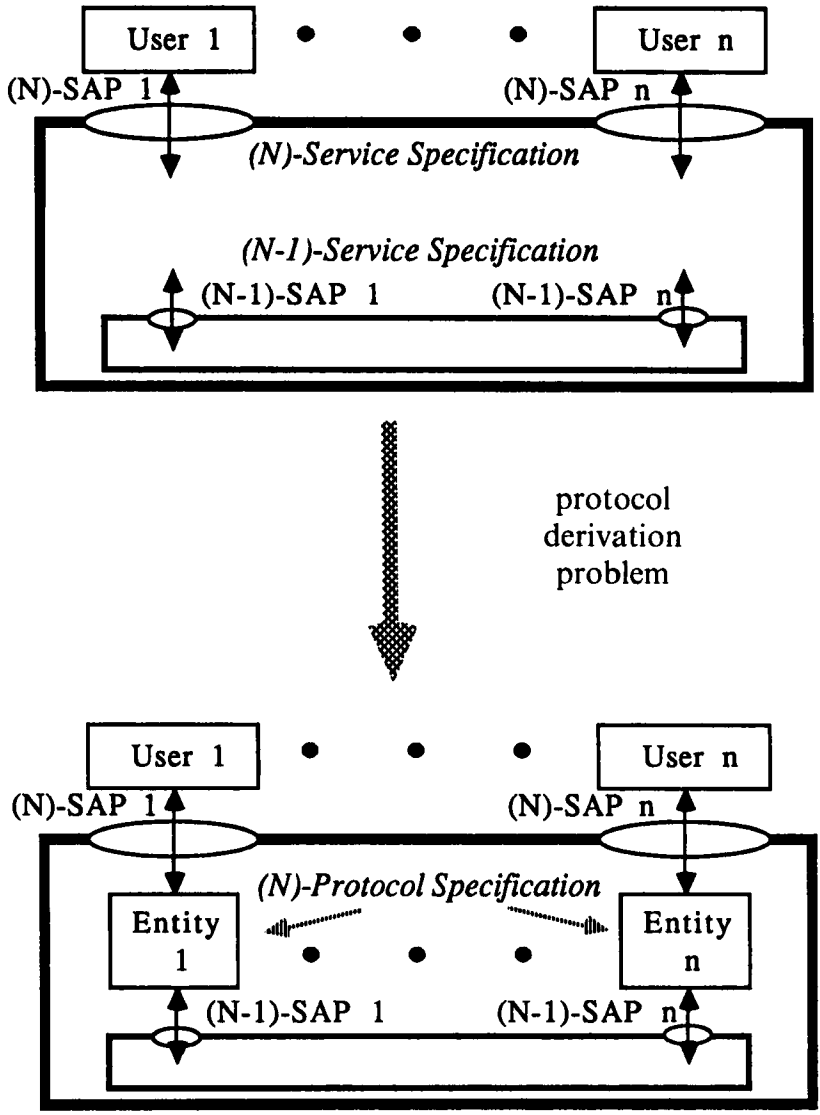


FIG. 19. From the (N)-service specification and the (N − 1)-service specification to the (N)-protocol specification

we want to find an algorithm for the protocol derivation problem. However, this protocol derivation procedure for an arbitrary communication service appears to be formidably difficult. As a result, we concentrate on a class of communication services whose behavior can be described by a set of directly coupled Finite State Machines (FSMs). This state-transition model allows the specifications of both terminating and nonterminating communication services. For a service specified in the state-transition model, we provide a protocol derivation algorithm that produces the protocol specification automatically once some further information about decision options and initiation options is given by the protocol designer. The provision of the above information is to make sure that the derived protocol specification is desired by the protocol designer.

6.2.1 The Model

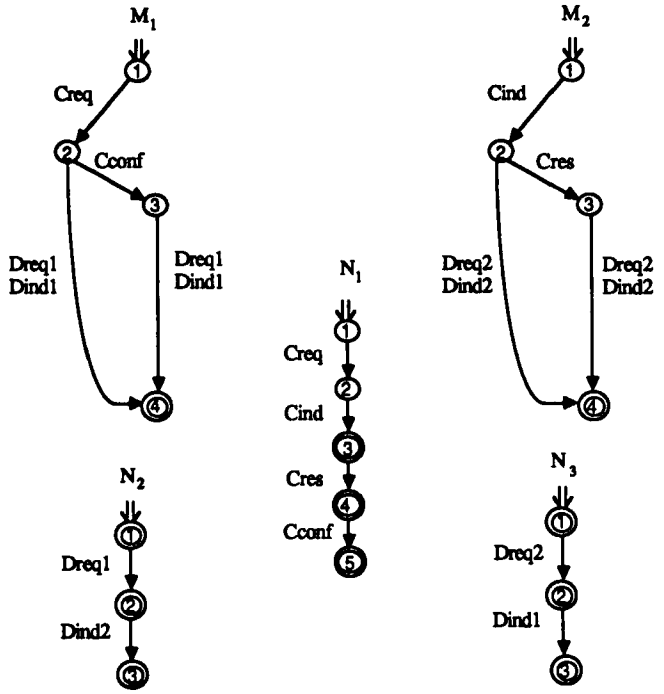
A service specification in our model (Chu and Liu, 1988a, 1988b) is composed of *local constraint FSMs* and *global constraint FSMs*, directly coupled with one another. One example is the connection establishment and release phases of the simplified ISO transport service, as specified using the modified Communicating Sequential Processes (CSP) of Liu and Liu (1984), without the provider-initiated disconnections. In this service specification (see Fig. 20) there are five service FSMs: two local constraint FSMs, M_1 and M_2; and three global constraint FSMs, N_1, N_2, and N_3.

Using a set of directly coupled service FSMs to specify a service may result in an inconsistent description; therefore, we provide an "inconsistency checking" algorithm to detect any inconsistency. An inconsistent nonterminating service specification is one that may *deadlock*, whereas an inconsistent terminating service specification is one that may reach a global state from which no final global state can be reached (called *improper termination*). The inconsistency checking algorithm actually constructs the reachability graph in which the deadlock (or improper termination) is checked.

A protocol specification consists of two entity specifications, each of which, similar to a service specification, contains local protocol FSMs and synchronizing protocol FSMs, directly coupled with one another.

6.2.2 Protocol Derivation Algorithm

In deriving a protocol specification from a given service specification, the local constraint FSMs of the service specifications can be embedded directly into entity specifications as the local protocol FSMs since local constraint FSMs perform decision locally without requiring any communication between entities. On the other hand, global constraint FSMs enforce the relative



Service primitive	*P*	*Sap(P)*
Creq	-- Connection request (from User 1)	SAP_1
Cind	-- Connection indication (to User 2)	SAP_2
Cres	-- Connection response (from User 2)	SAP_2
Cconf	-- Connection confirmation (to User 1)	SAP_1
Dreq1	-- Disconnect request from User 1	SAP_1
Dind2	-- Disconnect indication to User 2	SAP_2
Dreq2	-- Disconnect request from User 2	SAP_2
Dind1	-- Disconnect indication to User 1	SAP_1

⇒ : initial state pointer

◎ : final state

FIG. 20. The specification of the simplified ISO transport service

execution order of service primitives associated with different Service Access Points (SAPs), requiring protocol entities serving different SAPs to communicate with each other to synchronize the execution order of service primitives. The algorithm to derive the synchronizing protocol FSM pair (two synchronizing protocol FSMs, one for Entity 1 and the other for Entity 2) from a

global constraint FSM has three major steps:

1. Insert some intermediate transitions between service primitive transitions according to the specified decision option of a service state in a global constraint FSM.
2. Adjust the initial state pointer according to the given initiation option.
3. Project the resultant refined FSM onto Entity 1 and Entity 2 independently to produce the desired synchronizing protocol FSM pair.

6.2.3 *Error-Recovery Transformation*

To enable our algorithm to deal with erroneous underlying communication services, we further devise an error-recovery transformation procedure. The error-recovery transformation procedure consists of three transformation rules applicable to three different patterns of transitions in the synchronizing protocol FSM produced by the protocol derivation algorithm from a service specification.

A problem, called the *sink-state problem*, has been created by sink states of synchronizing protocol FSMs in the error-recoverable protocol produced by applying the error-recovery transformation to a protocol derived from the protocol derivation algorithm. The problem can be fixed by forcing an entity to send a "sink command" to the other entity once it reaches a sink state. This repair corresponds to another transformation working on the portions of an error-recoverable protocol specification that cause the sink-state problem.

The *duplicate acceptance problem* would result from applying the error-recovery transformation to the protocol produced by the protocol derivation algorithm. This problem can be resolved by performing another transformation on any error-recoverable protocol produced by the protocol derivation algorithm and the error-recovery transformation procedure.

In short, we have developed a protocol derivation algorithm, an error-recovery transformation procedure, and transformations to fix the sink-state problem and the duplicate acceptance problem, all of which are based on the state-transition model (Chu, 1989). Due to the space limitation these are omitted here.

As an example, let us apply the protocol derivation algorithm and the error-recovery transformation procedure to the simplified ISO Transport Service as shown in Fig. 20. We obtain the protocol specification shown in Fig. 21, where M_1 and M_2 are local protocol FSMs for Entity 1 and Entity 2, respectively, and the rest are synchronizing protocol FSMs for Entity 1 or Entity 2.

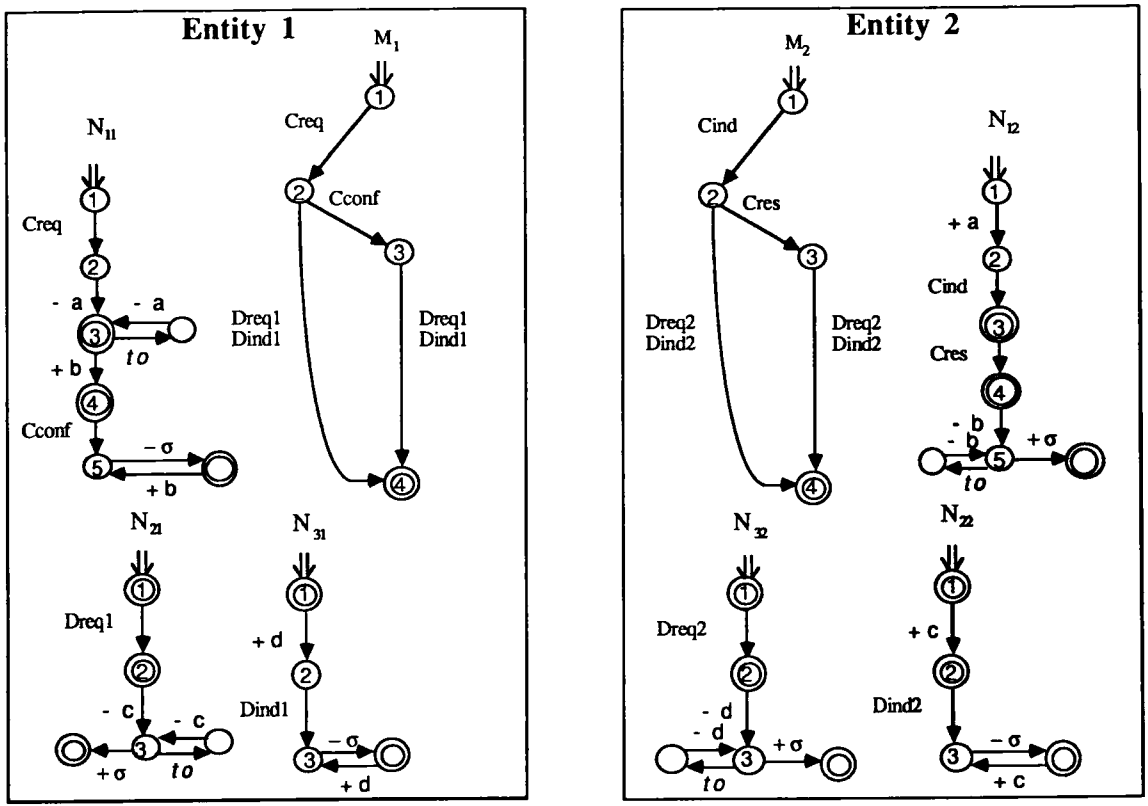


FIG. 21. Error-recoverable protocol for the simplified ISO transport service

6.3 Future Work

We would like to reemphasize that the right approach to protocol synthesis should be one that treats the service concept formally. For protocol synthesis using the service concept, the proposed protocol synthesis technique is our first attempt in this direction. However, there is more to be done in order to arrive at a truly satisfactory protocol synthesis technique. Several limitations in our approach have been identified and itemized as follow, and future work on removing the limitations are also discussed.

1. A high degree of concurrency in the execution of service primitives can be achieved in our model through service FSMs running in parallel, but synchronized by direct coupling of service primitives. Even so, any synchronizing protocol FSM pair produced by the protocol derivation algorithm is always closely synchronized in the sense that the communication pattern of the synchronization messages is "handshaking," there are no message collisions, and at most two messages are in transit at any instant for the synchronizing protocol FSM pair. Therefore, the expressive power of our state-transition model is still limited as far as the control aspect of protocols is concerned. The study of an appropriate way to enhance the expressive power of our model is in order.
2. For modeling real-life protocols, the addition of parameter, variable and time specifications to our service model is mandatory. However, the addition may have an extensive impact on the protocol derivation algorithm, requiring more careful investigation.
3. The optimization issue for communication protocols raised in papers by Bochmann and Gotzhein (1986), Merlin and Bochmann (1983), and Ramamoorthy and Dong (1982) is still an open question. The issue in our context for either error-free protocols or error-recoverable protocols is also a challenging work. Two points about the optimization of the generated error-recoverable protocol specifications are identified, i.e., the elimination of redundant timers and the use of negative acknowledgments. The discussion on them follows.

(a) Let us consider the optimization issue on a transformed error-recoverable protocol specification. If we make stronger the fairness assumption about the communication media, some timers may become redundant and thus be eliminated. For example, many versions of the Alternating Bit Protocol (ABP) only use a timer in the sender for retransmission of lost messages. But our error-recovery transformation would impose a timer in both the sender and the receiver. However, the fairness assumptions about the communication media in their specifications of the ABP and our specification are not the same. They assume that the communication media

will correctly deliver a message *infinitely often* if the message is retransmitted an infinite number of times. On the other hand, our assumption is that the communication media will correctly deliver a message *at least once* if the message is retransmitted an infinite number of times. Obviously, their assumption is stronger than ours, thus making the use of a timer only in the sender justifiable. In case we also make our assumption as strong as theirs, we should be able to remove the timer in the receiver without sacrificing the functionality of the protocol. At present, we still do not have a general solution for eliminating redundant timers from any produced error-recoverable protocol specification if the fairness assumption about the communication media is made stronger.

(b) The use of negative acknowledgments in an error-recoverable protocol may reduce the time period between two consecutive transmissions of the same message, thereby increasing the average throughput of message delivery between two service users. However, it also complicates protocols and introduces some processing overhead. We believe that the use of negative acknowledgments should depend on the actual environments in which the protocol will be implemented. In case we do wish to use negative acknowledgments in our error-recoverable protocol specifications, it is interesting to study the right way to include them in the specifications.

7. Timed Models and Performance Analysis

As discussed in Sections 3 and 4, various untimed formal models have been developed for protocol specification, validation and verification. However, these untimed models cannot be used to verify a protocol in which time constraints are essential for the correct functioning of the protocol. For example, Shankar and Lam (1982) found that in order to prove a desired timeout condition for a simple protocol, untimed modeling of that protocol is not adequate; Merlin and Farber (1976) discovered that in order to study recoverable protocols, a timed model (time Petri net) must be used to remove inherent limitations of the untimed model; Walter (1983) also found the inadequacy of untimed models when he tried to model and analyze a complex surveillance protocol for distributed systems; and more recently, Jain and Lam (1987) reported the necessity of timed protocol modeling when verifying a real-time protocol. It is worth noting that even the alternating bit protocol (ABP) used for illustration in Sections 3 and 4 is time-dependent should one remove the assumption that the medium cannot lose any message in transmission. In this case, the sender then has to employ a timer to do error recovery, and correct functioning of the protocol depends highly on the correlated time factors such as the timeout period, transmission delay, and processing speed of the entities. Furthermore, as a question raised by the title

of a paper by Yemini and Kurose (1982) ("Can current protocol verification techniques guarantee correctness?"), functional correctness is not the only concern in protocol design. Another indispensable aspect of the protocol is its performance; and as a matter of fact, the foundation should be, once again, *timed* models because without time specification performance analysis cannot be done in a formal model.

Therefore, there seems to be two main goals for timed protocol modeling. One goal is for verification of time-dependent protocols. The other goal is for performance analysis of protocols. But very few models are targeted for both. As pointed out by Yemini and Kurose (1982), there is indeed a need to provide a unified approach to the functional and performance analysis of protocols. It is also interesting to note that most effort in extending untimed models to timed models is for performance analysis of protocols.

In this section we first briefly survey various timed models that have been proposed in the literature. We then present several timed models we have developed for both protocol verification and performance analysis.

7.1 Previous Timed Models

In this section we present a brief survey of the time extensions done on various formal models described in Section 3: the CFSM model, the Petri nets model, the CCS model, the CSP model, and the Abstract Machine model.

1. *Timed CSFM Models.* Most work in this domain is done by researchers in the IBM Zurich Research Laboratory. Basically, there are three approaches to adding time specifications to the CFSM model. Two are done to predict performance of a protocol from its formal specification (Rudin, 1983, 1984; Kritzinger, 1984). The remaining one is done to verify a protocol modeled more realistically, namely by including time information of network components as part of the model (Bolognesi and Rudin, 1984).

2. *Timed Petri Nets Models.* Enormous work has been done on extending untimed Petri net models to timed models in order to model and analyze not only communication protocols but also other systems such as real-time and multiprocessor systems. Nevertheless, our major concern here is those models related to protocols. Those models differ according to how time is associated with the net and in what form. Three different terms have been used by various researchers: *Timed Petri Nets* (Zuberek, 1986; Garg, 1985; Walter, 1983; Razouk and Phelps, 1984; Holliday and Vernon, 1987), *Time Petri Nets* (Merlin 1976; Berthomieu and Menasche, 1983; Menasche, 1985), and *Stochastic Petri Nets* (Molloy, 1982; Marsan *et al.*, 1984; Zuberek, 1985).

3. *Timed CCS Models.* The timed model proposed by Nounou and Yemini (1984) is a timed CCS model even though they used a different set of notations. Basically, time information is not specified on the level of individual communicating entities, but on the level of the global behavior tree after all the communicating entities are combined by parallel composition. The global behavior tree captures all the possible interaction sequences and nondeterministic behavior of a protocol.

4. *Timed CSP Models.* A timed CSP model was proposed by Reed and Roscoe (1986) to verify real-time properties of communicating processes while retaining compatibility with the semantics of the original untimed model. It is an extension of Hoare's CSP trace model. Recently, Zic (1987) extended the time CSP model by incorporating *probability* specification in the model. This is done by associating probabilities with CSP's nondeterministic choice operators. The purpose is to allow both protocol performance specification and verification in timed CSP.

5. *Timed Abstract Machine Models.* Shankar and Lam (1982, 1984) have proposed a timed abstract machine model that uses discrete-valued timer variables to measure the elapse of the time and time events to age the timer. Those time variables and time events are local to each process in the model. Timer variables from different processes are uncoupled and can tick at different rates. Nevertheless, an ideal timer is assumed, based on which local timers are constrained within a specified error bound by the *accuracy axiom.*

As all the timed protocol models discussed above are extensions of the corresponding untimed models, the major issues on timed protocol modeling can be seen from two points of view, namely, in what form the time specification is represented and with which component of the model it is associated.

1. *In What Form The Time Specification Is Represented.* There are three possible forms of time specifications:

(a) Constant (deterministic) time.
(b) Time interval.
(c) Stochastic time (random variable, mostly with exponential distribution).

2. *With Which Component Of The Model It Is Associated.* There are two aspects of model components the time can be associated with: model in large or model in small.

(a) *Model in large.* When we look at a model in large, there are two ways to associate time specifications with the model. One way is to associate time with the components of individual communicating entities before they are composed together. The other way is to associate time with the components of the global protocol behavior after individual entities are composed together. Examples of the former are time specifications in timed Petri nets and timed abstract machines. Examples of the latter are time specifications in timed CCS.

(b) *Model in small.* When we look at a model in small, the ways to associate time specifications are quite model-dependent and, in fact, any natural association is possible. For example, there are four major components in a Petri net: place, transition, arc from a place to a transition, and arc from a transition to a place. Likewise, time can be associated with either states or transitions in the CFSM model, with events in CCS and CSP, and with state variables in an abstract machine.

Analytic power of the timed models can be discussed from two points of views: (1) what purpose it is used for, verification or performance, and (2) how it is related to its original untimed model. From this survey study, we have the following observations:

1. *Form of time specification.*

   (a) Stochastic time specification is only suited for performance analysis.
   (b) Time interval specification is best suited for verification, but its analytic method is hard to derive.
   (c) Constant time specification can be used for verification and performance analysis. But when used for verification, it is not as good as time interval specification; when used for performance analysis, it is not as good as stochastic time specification.
   (d) Moreover, probability specifications need always be brought into the model if performance analysis is going to be supported by non-stochastic time specifications such as constants or intervals.

2. *Time association with the model components.*

   (a) We believe that the association of time specification with individual communicating entities is more realistic and more convenient than with the global behavior of the protocol.

(b) How time is associated with the components of communicating entities strongly decides how difficult it will be to derive an analytical method. Thus it should be done with the analytic method in mind.

3. *Which untimed model is better for time extension.* It seems that time extension in one model can often be also applied to another model. Thus, on which untimed model the time extension is done is not as important as other issues in the current research. One of these issues is the lack of a timed model that can support both verification and performance analysis of protocols.

In summary, there is no doubt that time modeling choices made in a timed protocol model greatly affect the analytic power it can provide. Normally, there is a tradeoff between modeling and analytic power and it is not easy to reach a balance point when making these choices. In the following two subsections we briefly describe several timed models we have developed in the past.

7.2 TTG and TTG$^+$ Models

The TTG model (Lu, 1986) is an extension of the untimed TG model (Teng, 1980) for the verification of time-dependent and time-independent, synchronous and asynchronous protocols. The extension was done in several aspects.

1. The specification is divided into two classes, namely *entities* and *channels.* They are component grammars modeling the communicating entity and the medium connecting them, respectively.
2. Time specifications are added to the model. There are three kinds of time information that can be specified in the model, namely *timeout interval, rule firing time,* and *transmission delay.* The rule firing time is associated with each *production rule of the entities* to denote the delay and execution time of the rule. The transmission delay is associated with each *production rule of the channels* to denote how long a message will take to reach its destination. The timeout interval is specified when a *timer* is activated.
3. Two more actions for setting and clearing timers are introduced to make the notion of timers explicit. They are for activating and deactivating a timer. Within an entity, there is a timeout handler for each timer used, which specifies the service actions taken when the timer expires.

Each time specification in the TTG model is in the form of an *interval* $[t_{\min}, t_{\max}]$, where $t_{\min}$ is a nonnegative integer, $t_{\max}$ is a nonnegative integer or ∞

(infinity), and $t_{max} \geq t_{min}$. When specified as a rule firing time, t_{min} is the minimum delay time before the actions in a production rule can be executed after the rule is *enabled*, and t_{max} is the maximum elapsed time before which all the actions in an enabled rule must be completed. When specified as a timeout interval or as a transmission delay, t_{min} and t_{max} are the minimum and maximum times required for a timer to expire and for a message to reach its destination, respectively.

Based on such an extended timed model, Lu (1986) derived a reachability analysis algorithm to verify various properties of a protocol modeled by TTG. The algorithm has been mechanized and used successfully to validate the ABP, the X.21 and the IBM token-ring protocol (see Section 10.6).

Though the TTG model developed by Lu (1986) is capable of handling time-dependent protocols, it can be applied to only a restricted class of protocols. To overcome this drawback, we have modified the TTG model and developed a novel algorithm of timed reachability analysis based on the new model (Lin, 1988). In order to distinguish the new model from the old TTG model, it is called the TTG^+ model.

7.3 ITTG Model

In this section, we present an integrated approach to verifying general properties of protocols and to analyzing their performance based on a formal model called Integrated Time Transmission Grammar (ITTG). It is an extended and refined protocol model resulting from an evolutionary series of the Transmission Grammar-based models: the TG (Teng, 1980), the TTG (Lu, 1986), and the TTG^+ (Lin, 1988), and ETG (Chu, 1989) models. Basically, the extension on the TG-based models follows the same line of evolution as other models, i.e., from untimed to timed modeling of protocols. But here we move one step further to integrate two major purposes of timed protocol models, viz., verification and performance analysis, in a single framework. The major extension done to previous timed models (TTG and TTG^+) is the incorporation of time specifications into the models. Now in order to facilitate performance analysis also, both time and probability specifications are incorporated into the ITTG model. Figure 22 shows the relation between these TG-based models.

Basically, the ITTG (Lin and Liu, 1988b) model is a set of regular grammars consisting of three distinct parts: *entities, channels*, and *timeout handlers*. Each of the entities, channels, or timeout handlers is a regular grammar or a set of regular grammars. Together they specify a communication protocol elegantly. Non-terminals and terminals in ITTG are referred to as *states* and *actions* respectively in the following.

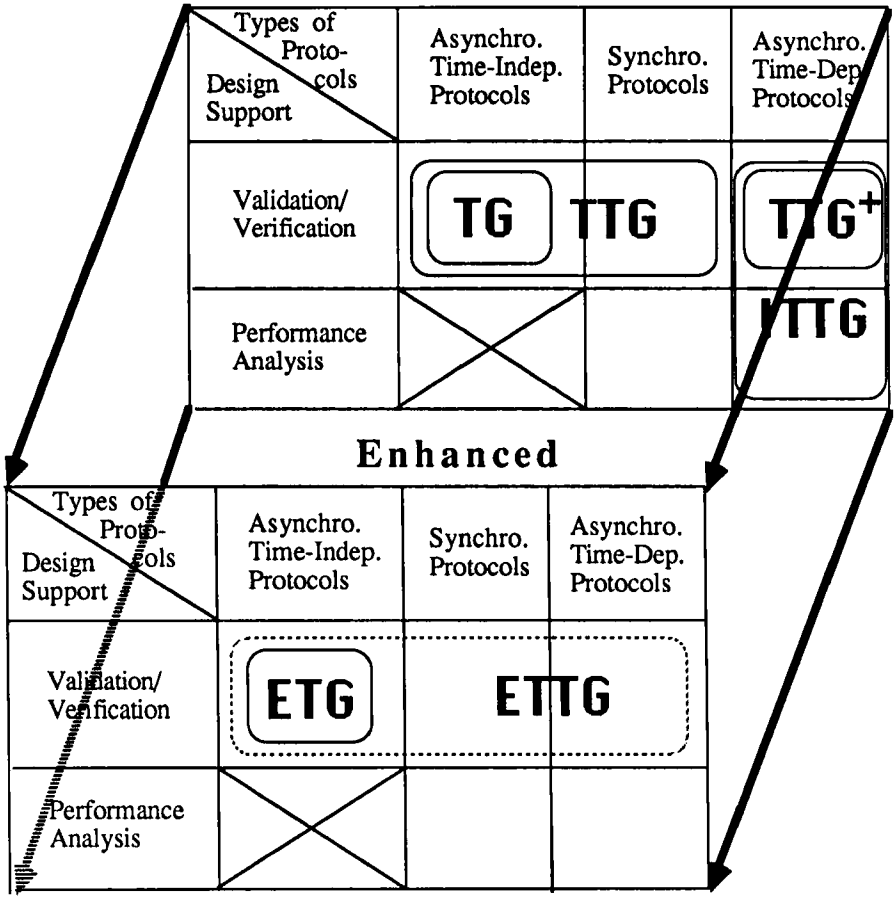


FIG. 22. Family of the transmission grammar-based models

1. *Entities in the ITTG Model.* An *entity* in ITTG is a regular grammar preceded by "$entity *entity-id.*" The actions performed by an entity are the following:

Q.*entity-id*[.*entity-id*....].*message-name*
D.*entity-id.message-name*
S.*timer-id.time-interval*
C.*timer-id*
internal-action-name

where Q and D specify communicating actions corresponding to sending (enQueueing) and receiving (Dequeueing) of a message; S and C are timer actions to Set and Clear a timer; and *internal-action* specifies those operations invisible to other entities.

Each production rule of an entity is of the form:

⟨*current-state*⟩.[*time-interval*] ::= [*probability-value*]*action-1*
[*action-2*...⟨*next-state*⟩.

where *action*-1 is either a Q, a D, or an internal action, but any action that follows must be a timer action. The ⟨*current-state*⟩ that appears in the first production rule of the grammar is implicitly defined as the *initial state* of the entity. Usually, a state may have more than one production rule. In ITTG, these rules are grouped together and each of them is separated from others by a comma (",").

A state is said to be *passive* if none of its production rules contains a Q or an internal action; otherwise, it is *active*. For an active state, a *time interval* must be specified and associated with it. Semantically, this time specification gives the minimum and maximum delays that an active state must be held before it can move to the next state. Conversely, for a passive state no time interval need be specified because how long a passive state is held will be decided by other external events. An active state with more than one production rule containing a Q or an internal action is called a *decision state*. For a decision state a *probability value* must be specified for each of its production rules containing either a Q or an internal action, such that the sum of all the probability values assigned to these rules is 1.

2. *Channels In The ITTG Model.* A *channel* in ITTG is a regular grammar preceded by "$channel [*entity-id-1*-⟩ *entity-id-2*] of size *number*.", which specifies not only the source and destination of the medium but also the capacity of the medium. Unlike an entity, a channel has only one state, called the *idle state*, and its possible actions are the following:

T.*in-message-name*
L.*in-message-name*
U.*in-message-name*
G.*in-message-name.out-message-name*

where T denotes correct message Transmission through the medium; L denotes message Loss in the medium; U denotes message dUplication (from one to two) by the medium; and G denotes that message *in-message-name* has been Garbled during the transmission and the garbled message delivered by

the medium is *out-message-name*. In short, these actions specify possible behaviors of a channel. For convenience of specification, if a medium has no discrimination against the message type, message name "*" is used to indicate *any* message sent through the medium.

The production rule of a channel is of the form:

⟨*idle-state*⟩ ::= [*probability-value*][*time-interval*] *action* ⟨*idle-state*⟩.

where *action* is either a T, L, U, or G and *time-interval* specifies how long it will take for a message to get through the medium. Note that in order to accurately estimate channel busy time, a time interval is specified even for the message lost in the medium. A channel with multiple production rules for an *in-message-name* is called an *unreliable channel*. For an unreliable channel a *probability value* must be specified for each alternative production rule of the *in-message-name* such that the sum of the probability values assigned to these rules is 1.

3. *Timeout Handler in the ITTG Model.* A *timeout handler* in ITTG is a set of regular grammars preceded by "$timeout-handler *timer-id* of entity *entity-id* for (*message-name, acknowledge-name*) in channel *entity-id* = *entity-id*, [(*message-name, acknowledge-name*) in channel *entity-id* = *entity-id*...].", where both the entity and the messages/acknowledgments served by a particular timer are specified. Each grammar in the set consists of *only one* production rule.

Unlike an entity or a channel, each production rule of a timeout handler refers to the state of the entity that the handler serves rather than the state of the handler itself; as such it carries different semantics. The form it takes is as follows:

⟨*current-state*⟩ ::= *time-interval action-1* [*action-2* ...] ⟨*next-state*⟩.

where ⟨*current-state*⟩ must be unique for each timeout handler and a *time-interval* must be specified to indicate the time taken to execute this timeout service.

The actions that can be performed by a timeout handler are Q.*entity-id*[.*entity-id*....].*message-name* and S.*timer-id.time-interval*, which model the timeout-retransmission mechanism normally employed in communication protocols. Seemantically, the rule specifies what kind of service should be done when a timeout occurs due to a certain timer and in a certain state of the entity.

Based on the ITTG model, the techniques for both protocol verification and performance analysis have been developed (Lin, 1988). Basically, verification of a protocol is done based on the properties of both reachable states and their reachability graph. On the other hand, performance analysis of a protocol is done based on the extraction of timed probabilistic (TP) graphs from the

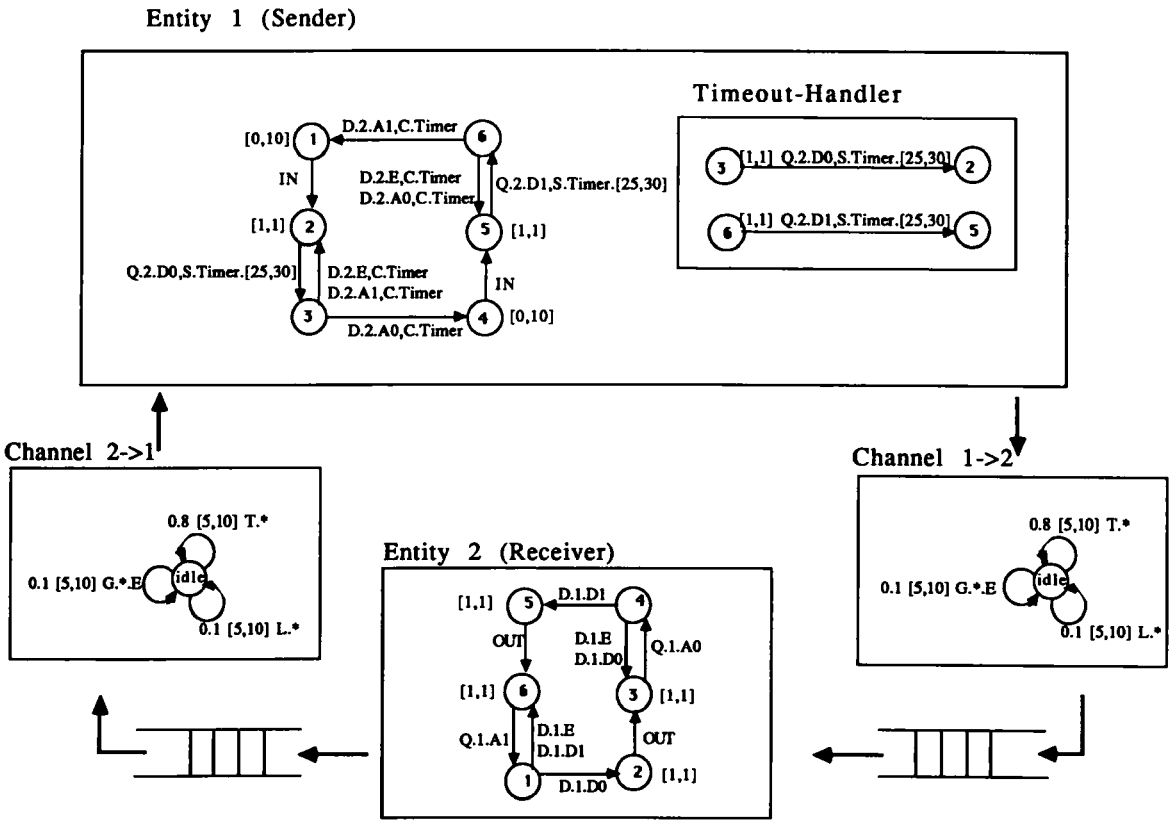


FIG. 23. ITTG state diagram of the alternating bit protocol

```
$entity 1.
<1>.[0,10] ::= IN <2> .
<2>.[1,1]  ::= Q.2.D0, S.Timer.[25,30] <3> .
<3>        ::= D.2.Er, C.Timer         <2> ,
               D.2.A1, C.Timer         <2> ,
               D.2.A0, C.Timer         <4> .
<4>.[0,10] ::= IN <5> .
<5>.[1,1]  ::= Q.2.D1, S.Timer.[25,30] <6> .
<6>        ::= D.2.Er, C.Timer         <5> ,
               D.2.A0, C.Timer         <5> ,
               D.2.A1, C.Timer         <1> .

$timeout-handler Timer of entity 1 for (D0,ACK0)
          in 1=2, (D1,ACK1) in 1=2.
<3>        ::= [1,1] Q.2.D0, S.Timer.[25,30] <2> .
<6>        ::= [1,1] Q.2.D1, S.Timer.[25,30] <5> .

$entity 2.
<1>        ::= D.1.D0 <2> ,
               D.1.Er <6> ,
               D.1.D1 <6> .
<2>.[1,1]  ::= OUT    <3> .
<3>.[1,1]  ::= Q.1.A0 <4> .
<4>        ::= D.1.Er <3> ,
               D.1.D0 <3> ,
               D.1.D1 <5> .
<5>.[1,1]  ::= OUT    <6> .
<6>.[1,1]  ::= Q.1.A1 <1> .

$channel 1->2.
<idle>     ::= 0.8 [5,10] T.* ,
           ::= 0.1 [5,10] L.* ,
           ::= 0.1 [5,10] G.*.Er .

$channel 2->1.
<idle>     ::= 0.8 [5,10] T.* ,
           ::= 0.1 [5,10] L.* ,
           ::= 0.1 [5,10] G.*.Er .
```

FIG. 24. ITTG model of the alternating bit protocol

global reachability graph. The final measures of protocol performance can be represented in the form of an interval, indicating the performance parameters of the protocol under the best and the worst cases. Due to space limitation, those techniques are omitted here.

As an example, let us consider the alternating bit protocol (ABP) discussed in Section 2. Unlike the ABP described in Section 3, the ABP illustrated here takes into consideration the fact that the medium may lose a message in transit. Figure 23 shows the state diagram of this more realistic ABP. Figure 24 lists the formal specification of the protocol in ITTG. After performing the reachability analysis, the ABP is found to be free from all erroneous protocol properties such as unspecified reception, unspecified timeout service state, deadlock, channel overflow, improper timer action, and premature timeout. Nevertheless, four tempo-blocking cycles are identified: $29 \rightarrow 27 \rightarrow 28 \rightarrow 39$, $9 \rightarrow 23 \rightarrow 13 \rightarrow 14 \rightarrow 8 \rightarrow 9$, and $14 \rightarrow 16 \rightarrow 17 \rightarrow 14$.

Once logical correctness of the protocol is verified, the next step is to compute performance measures of the protocol based on the global state graph already available after the verification. First, the TP graphs in the best and the worst throughput cases are extracted. Then, based on the extracted TP graphs we get the following performance measures after computation.

Channel Utilization	
$1 \rightarrow 2$:	[0.291971, 0.299774]
$2 \rightarrow 1$:	[0.294102, 0.301963]
Throughput:	[0.021607, 0.044370]
Efficiency:	[0.571880, 0.634422]

The details of performance computation for the ABP can be found in (Lin, 1988). Basically, the ABP specified here can transfer from 21.7 up to 44.4 messages per second if one time unit is equal to 1 msec. Both channel utilizations are approximately 30% without much difference under the best and the worst cases, and about 60% of the time the protocol is doing something effective.

8. Protocol Conversion

As discussed earlier, users on different computer networks cannot easily communicate with each other due to the proliferation of different network architectures and communication protocols. *Protocol conversion* is to resolve the incompatibility between protocols so that users on different networks can communicate with each other. So far, most protocol converters have been

constructed manually with ad hoc approaches due to the lack of a formal theory for protocol conversion (Green, 1986). Thus, protocol conversion is the most recently established area in protocol engineering (Rudin, 1988).

In this section, previous work on the development of a formalism for protocol conversion by other researchers is first presented. Then our effort in developing a formalism, which is more powerful in modeling protocol conversion and requires less human ingenuity, is discussed. Finally, we point out possible directions for future research.

8.1 Previous Work

To the best of our knowledge, there are only two major formalisms that have been proposed for protocol conversion by other researchers. We will examine each of these two approaches in more detail.

Okumura's Model. In the model proposed by Okumura (1986), a protocol is modeled as a tuple $\langle A_0, \ldots, A_n \rangle$ of Communicating Finite State Machines (CFSMs) with message set M, where M is the union of the set M_{ij} of messages from A_i to A_j $(i, j = 0, \ldots, n)$, and each set M_{ij} is mutually exclusive. A four tuple $A_i = (\sigma_i, M_i^{\pm}, \delta_i, q_i)$ is a CFSM which contains the following components:

1. A non-empty finite set δ_i.
2. A finite set $M_i^{\pm}$.
3. A partial function δ_i from $\sigma_i \times M_i^{\pm}$ to σ_i.
4. A designated element q_i in σ_i.

Suppose we have two protocols A and B, both of which contain two CFSMs:

$$A = \langle A_0, A_1 \rangle \qquad B = \langle B_0, B_1 \rangle.$$

Figure 25 is an example of protocols A and B, where A_1 and B_1 transmit messages to A_0 and B_0, respectively. Protocol A is typical of the polling model, whereas protocol B is typical of the ack-nack model. Each state is denoted by a circle and each transition is denoted by an edge. The symbol $-m$ on the edge means "send message m" and $+m$ means "receive message m."

The goal of the conversion is to allow the communication components of one architecture to communicate with those of another architecture. In Fig. 25 CFSMs A_0 and B_1 are assumed to be the components to communicate with each other. A CFSM C is put between A_0 and B_1 to interpret messages exchanged between these two components.

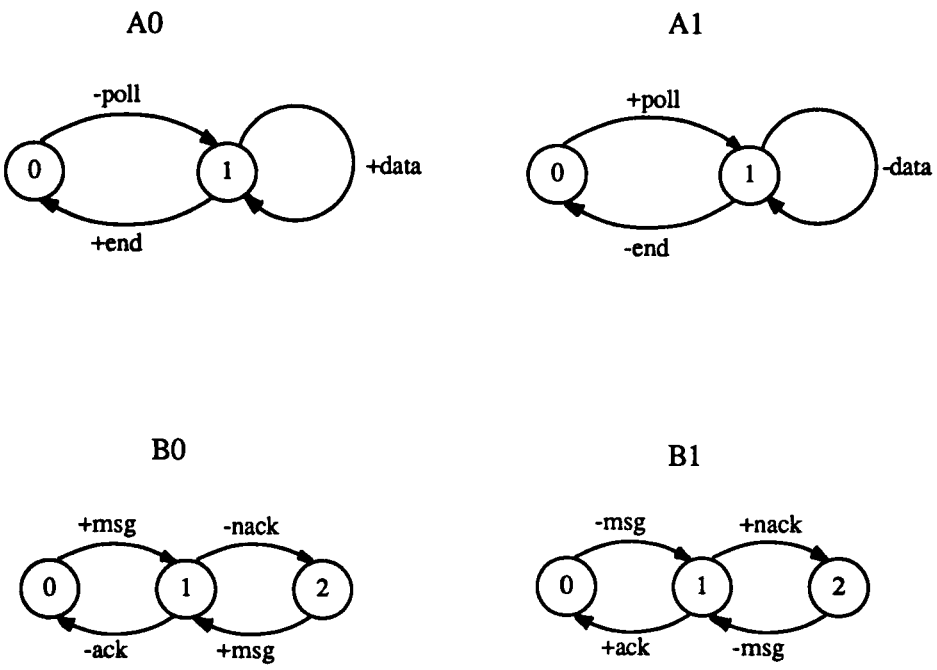


FIG. 25. Examples of two types of protocols
Protocol A is polling type protocols
Protocol B is ack-nack type protocols

In protocol $\langle A_0, C, B_1 \rangle$, A_0 communicates with (C *and* B_1) as if it communicates with its original partner A_1, and B_1 communicates with (A_0 *and* C) as if it communicates with its original partner B_0.

Furthermore, Okumura defined *external equivalency*, which guarantees a similar environment for CFSM A_0 (B_1) to the original protocol $\langle A_0, A_1 \rangle$ ($\langle B_0, B_1 \rangle$). Assume a protocol $P = \langle A_0, C, B_1 \rangle$ is given, and protocol P satisfies external equivalency iff for any executable sequence α, the subsequence of α, $\alpha|_A(\alpha|_B)$, which contains only the messages in message set M_A (M_B) of CFSM A (B), is also executable in A (B).

In solving the problem of how to decide the appropriateness of exchanging messages from a semantic viewpoint, Okumura proposed using the *conversion seed*. She assumes that the rule for the occurrence of a significant message is written in the form of a regular language and can be defined by an automaton. Since how precisely the functions in one protocol will be interpreted in terms of the functions in the other protocol may depend upon the design objectives,

the conversion seed should be given by the protocol converter designer after carefully studying the two protocols to be converted.

In Okumura's approach, conversion seed K is used to describe how the protocols are converted. The conversion seed $K = (\sigma_K, M_K, \delta_K, q_K, F)$ is an automaton over a significant message set $M_K \subseteq (M_{A1}^{\pm} \cup M_{B0}^{\pm})$ with final state $F = \delta_K$, and gives the guidelines and properties for protocol converter generation.

Given the conversion seed, the message sequences in the newly constructed protocol $P = \langle A_0, C, B_1 \rangle$ can further be constrained to the ones that are really meaningful to the protocol converter designer. Since not all the message sequences accepted by $A_0(B_1)$ is accepted by conversion seed K, Okumura restricts that given any message sequence $\alpha \in L(P)$, $\alpha|_K$ should be also accepted by conversion seed K. This property is called *semantics equivalency*.

According to the previous arguments, Okumura further defines that a CFSM C can be called a protocol converter for the given protocols $A = \langle A_0, A_1 \rangle$ and $B = \langle B_0, B_1 \rangle$ with conversion seed K iff protocol $\langle A_0, C, B_1 \rangle$ satisfies the following conditions:

1. External equivalency.
2. Semantic equivalency.
3. Freedom from unexpected input.
4. Freedom from deadlock.

The existence of the protocol converter is also proved to be decidable for the given protocols A, B and conversion seed K.

Some theorems proved by Okumura state that given CFSMs $A = \langle A_0, A_1 \rangle$ and $B = \langle B_0, B_1 \rangle$, which are deadlock free and unexpected input free, and conversion seed K; if there exists a converter C for $P = \langle A_0, C, B_1 \rangle$ with K, then there exists a converter D which is a sub-CFSM of $(A_1 \cdot B_0) \times \bar{K}$, where the $\cdot$ operator denotes the arbitrary shuffle operation (Teng, 1980) and the $\times$ operator denotes the intersection operation. CFSM $\bar{K}$ is the extension of CFSM K, and extends K's message set from M_K to $M_A \cup M_B$. The transition function of $\bar{K}$ is given as:

$$\delta : (\sigma_K, M_A \cup M_B) \to \delta_K,$$

$$\delta(s, m) = \begin{cases} \delta_K(s, m) & \text{if } m \in M_K \\ s & \text{if } m \notin M_K \end{cases}$$

With this upper limit $(A_1 \cdot B_0) \times \bar{K}$, Okumura proposed two protocol construction rules to construct the protocol converter: one is the subtractive approach from $(A_1 \cdot B_0) \times \bar{K}$, and the other is the additive approach from a null CFSM.

Though Okumura did propose the construction algorithm for the protocol converter, it is not satisfactory. For example, in Fig. 25, if we select A_1 and B_0 instead of A_0 and B_1 as components of protocols to talk to each other, then we would not be able to construct a protocol converter using this approach.

Lam and Calvert's Model. Formal techniques are also proposed by Lam (1986, 1988) and by Calvert and Lam (1987) to address the protocol conversion problem. Their approach makes use of *protocol projection*, an abstraction technique for verifying properties of complex protocols. The basic idea of *projection* is that a property of the complex system can be proved by finding a *property preserving transformation* to a simpler system, and by proving the property of the simpler system. The *image protocol* preserves the semantics of the original protocol.

Given a protocol A, a protocol projection is defined by partitioning the state spaces of each of A's processes. The idea is that process states that are to be functionally equivalent in the image protocol are aggregated into the same partition, and are mapped into the same *image process state.* Every message (event) of the original protocol either maps into a message (event) in the image protocol or has a *null image.* If the projection further meets some additional requirement, then the image is said to be *well-formed* and the image of any fair computation of the original protocol is also a fair computation of the image protocol.

If two protocols can be projected onto the same image protocol, then they share the inverse image of the safety properties of that image. Furthermore, if the image protocols are well-formed, then they have their safety and liveness properties in common. Based upon this idea, Lam and Calvert further proposed the following approach to solve the problem of protocol conversion.

First the properties required of the conversion are specified; then a projection of these two protocols onto a common image with the desired properties is looked for. If such an image protocol is found, then the job is done since we know that the protocols with the same image are semantically equivalent.

If the protocols do not have a common image with the desired properties, then a protocol converter has to be constructed. The candidate protocol converter can be obtained by considering the properties required and the structure of the processes involved in the conversion. If the candidate can be projected onto each of the original protocols, the inverse images of their properties are properties of the protocol converter. With this characteristic, Lam and Calvert claim that the safety and liveness properties (correctness) of the protocol converter constructed can be proved.

However, their formalism requires a careful study of the nature of the protocols to be converted. Also, the properties of the protocols should be well understood. Thus, a lot of human ingenuity is involved.

8.2 Our Conversion Approach

Our research effort is mainly concerned with automatic generation of protocol converters using a state-transition model. More specifically, we are interested in generating protocol converters for protocols specified in the Communicating Finite State Machine (CFSM) model. Due to the formidable difficulty and complexity of the problem, we are only concerned with a specific category of protocols, namely, two-entity nonterminating protocols. Given two protocols in this category along with the specification of the message sequence translation between these two protocols, a reception-error-free protocol converter can be generated with our proposed algorithm. Furthermore, if more related information is specified on these two target protocols, a deadlock-free protocol converter can be obtained.

The specification of how the translation between message sequences of the target protocols should be performed is accomplished by a set of CFSMs called the mapping CFSM set. Each CFSM in this set specifies the mapping between some message sequences of the two target protocols. Multiple CFSMs give the designer the capability to specify the mapping sequences that are independent. Moreover, each CFSM can be used to specify not only the mapping sequences but also the order of the mapping when ordering is critical. Semantically, all CFSMs in the mapping CFSM set are ORed together to establish the relation between the message sequences of two target protocols.

The process of deriving a protocol converter from two target protocols can be divided into four phases. In the first phase of the algorithm, a Universal Converter (UC) is constructed. A UC allows a sequence of one protocol to be mapped into any sequence of the other protocol and *vice versa*. Notice that no ordering restriction is imposed on these mappings. This UC can be constructed through an operation called Arbitrary Shuffle (Teng, 1980; Okumura, 1986). Arbitrary Shuffle allows a sequence of two CFSMs to be interleaved in any order (operator · performs the same operation). By taking one entity from each protocol and performing Arbitrary Shuffle between them, the resulting CFSM is a UC for these two protocols (see Fig. 26).

In the second phase of the algorithm, the mapping CFSM set is combined with the UC. The mapping CFSM set restricts the sequences allowed by the UC. During this process a reception error and/or a deadlock error may occur due to the logical conflict between the target protocols and the mapping CFSM set. Therefore, reception-error states are identified in this phase. In

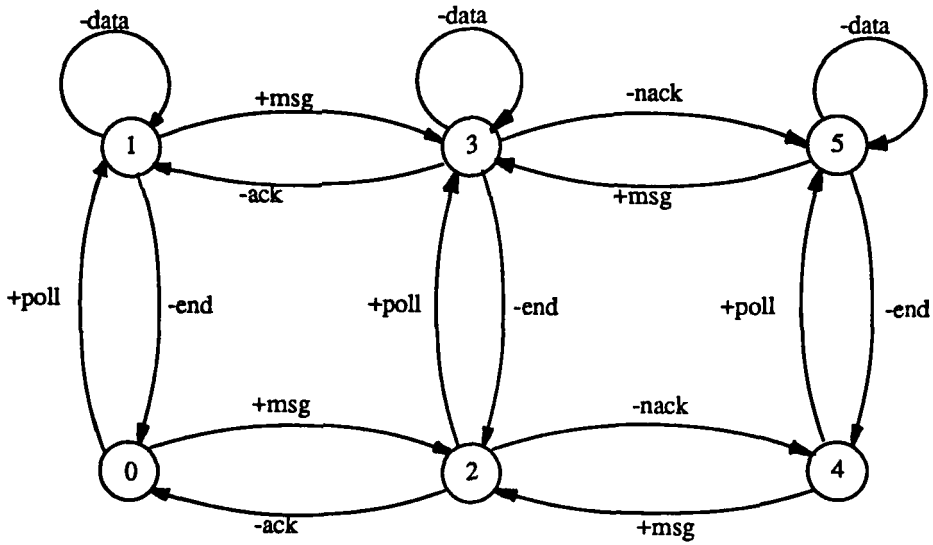


FIG. 26. A1 arbitrary shuffle with B0

the third phase of the algorithm, deadlock states can be recognized with more information concerning the two target protocols. Finally during the last phase, all reception-error states and deadlock states are removed to create a correct protocol converter.

In order to combine the UC with the mapping CFSM set, a CFSM Protocol Converter (PC) is created. Each state of the PC is labeled with a state of the UC and a state matrix of the mapping CFSM set. Each state in the matrix denotes the state of a CFSM in the mapping CFSM set. We call them the current state of the UC and the current state matrix of the mapping CFSM set. Accordingly, three rules are used for PC state transition:

1. *Transition firing.* Given a state of the PC, if a transition at the current state of the UC matches a transition at a state in the current state matrix, a new state of the PC is generated. The new state is labeled with the next state of the UC and the next state matrix of the mapping CFSM set.
2. *Regeneration of mapping CFSM.* Given a state of the PC, if a transition (T) at the current state of the UC does not match any transition at any state of the current state matrix, but T is a transition from the initial state of some mapping CFSMs, then this mapping CFSM is regenerated. Also, a new state of the PC is created and labeled with the next state of the UC and the next state matrix of the mapping CFSM set with the newly generated CFSM added to that set. For each new mapping CFSM regenerated, one new state of the PC is created.

3. *Removal of mapping CFSMs.* Given a state of the PC, if the current state matrix contains more than one copy of the same mapping CFSM (this is a result due to rule 2), and if any of those mapping CFSMs are in their initial states, then they can be removed from the current state matrix. This rule allows states to be removed once a regenerated mapping CFSM moves back to its initial state.

At a state of the PC, if a receive transition (T) at current state of the UC cannot find any identical transition at any state of the current state matrix, and T is not a transition from the initial state of any mapping CFSM, this state of the PC is labeled as a reception error state and no new states and transitions are generated from it.

To prevent rule 2 from repeatedly regenerating mapping CFSMs and causing the algorithm to run infinitely, a containment relationship between states is defined. A state X of the PC contains another state Y of the PC if they are labeled with the same current state of the UC, and Y's current state matrix is a submatrix of X's current state matrix. If a state X of the PC to which rule 2 is applied (called a regeneration state) contains an ancestor regeneration state, X is defined as a loop state and no new states and transitions are to be generated from state X. This criterion allows the algorithm to detect a loop of regenerating mapping CFSMs and to discontinue the process when it happens.

The third phase of the algorithm is for deadlock detection. Since deadlock states may be created in the second phase of the algorithm, we need a method to detect whether deadlock states exist and if they do, in what states the deadlock states are. To achieve this purpose, a critical send transition state list is needed for the target protocols. A critical send transition state is a state that if all the send transitions are removed from it, that state becomes a deadlock state. During the process of constructing the UC with arbitrary shuffle between the two target protocols, critical send transition states can be identified on the UC. States in the PC that are labeled with a critical send transition state of the UC are critical send transition states of the PC. By examining all the critical send transition states of the PC, we can decide what states are the deadlock states of the PC.

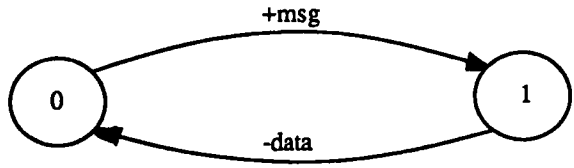


FIG. 27. Mapping CFSM set of the example in figure 25

Finally, the last phase of the algorithm removes all reception-error states, loop states, and deadlock states created by the previous two phases. All transitions in and out of these states are also removed. However, this process may create more deadlock and reception-error states. Repeatedly removing the error states eventually yields a correct protocol converter.

Applying the above algorithm to the example shown in Fig. 25 results in the protocol converter shown in Figs. 28 and 29. Figure 27 shows the mapping CFSM set for the example in Fig. 25. In this example, there is only one CFSM in the set. Figure 28 shows the protocol converter after the second phase of the algorithm, whereas Fig. 29 shows the correct protocol converter.

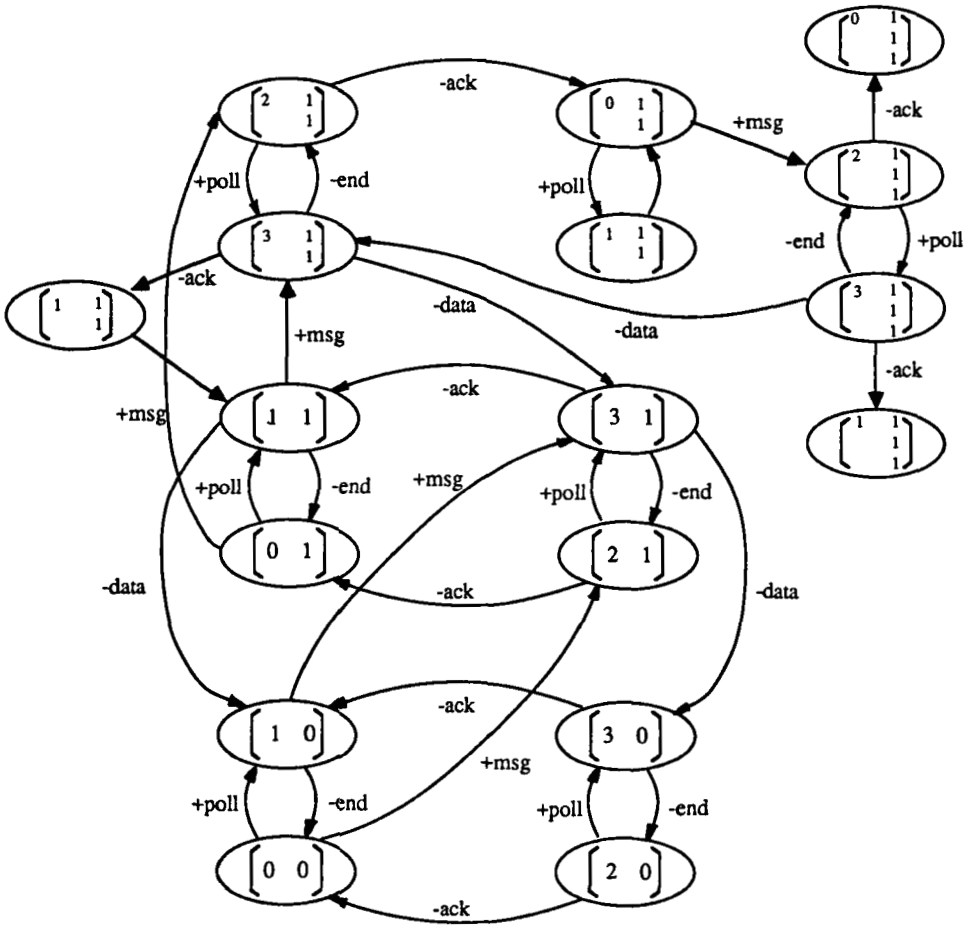


FIG. 28. Protocol convertor after second phase of the algorithm

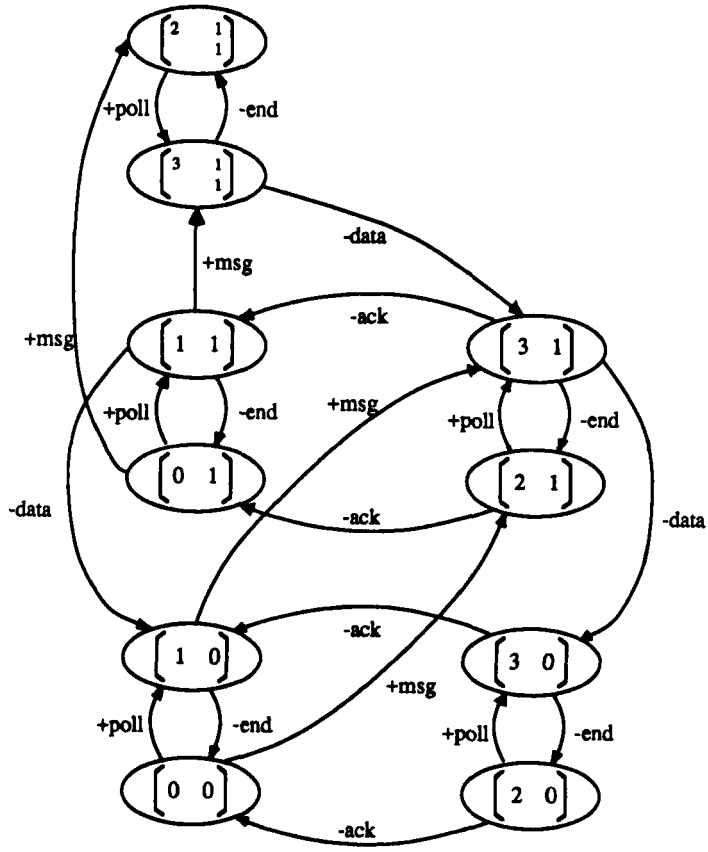


FIG. 29. The completely correct protocol convertor

The CFSM model has been used to model the ordered behavior of communicating protocols successfully in the past. Our work suggested a parallel CFSM model to describe relations between two protocols. This parallel model is powerful enough to model behaviors between protocols and yet it gives the designer the flexibility to specify either order or non-order relations between sequences of the protocols. We have developed an algorithm that can automatically generate a protocol converter according to the model and it is guaranteed to follow all the restrictions specified by the mapping CFSM set. However, the algorithm we have developed will not guarantee the existence of a protocol converter. An empty converter may be created if the logical conflict between the target protocols and the mapping CFSM set is very severe. More theoretical work is needed to determine when a converter cannot be found.

We are currently studying the possibility of incorporating priority into each mapping CFSM. This extension allows the protocol converter to put priority among different message sequences and enables the protocol converter to be constructed between the two target protocols with message priority and the protocols without message priority.

8.3 Future Work

Network interconnection has been studied by researchers for many years. From their work we know that a protocol converter can be implemented in two different architectures for network interconnection. One is the single gateway approach. The algorithm proposed in the previous section suits this model well. The other architecture model is the half gateway approach. To connect two networks, a node called a half gateway is inserted into each network. All messages from one network destined for the other network have to go through the half gateway. The half gateway performs the translation of messages and delivers the messages to its corresponding half gateway on the other network. This architecture is more efficient than the single gateway architecture since translation can be performed simultaneously. However, how to design a protocol converter to fit this architecture is a challenging task.

One possibility is to use the algorithm proposed above to construct a protocol converter for single gateway architecture. The protocol converter can then be partitioned into two half protocol converters, one for each half gateway. Partition of the protocol converter can be achieved by projection according to the message types (messages for protocol *A* or messages for protocol *B*). Synchronization messages then need to be added to each half protocol converter to synchronize them so their combined behavior is the same as the whole protocol converter. Synchronization messages in this model serve two purposes. Firstly, they synchronize the two half gateways to make sure they act properly, and secondly, actual information is also delivered between the half gateways through them. Other approaches for creating half protocol converters are also possible. It will be interesting to see if half protocol converters can be generated without going through the Universal Converter construction process.

The area of protocol conversion actually is only part of a more general area called protocol interworking. There are three types of protocol interworking behavior:

1. *Protocol conversion.* Gateways are inserted between networks. Protocol converters are implemented on those gateways to translate messages between different protocols. Users on different networks still

use the same access protocols to establish connection and transmit information to and from the network they are connected to. Insertion of gateways and protocol converters is transparent to them.

2. *Protocol overlap* (Lin and Liu, 1988a). Protocols of one network (protocol *A*) is modified to absorb protocols of the other network (protocol *B*). Protocol *B* executed under one phase of protocol *A*. Users on network *A* first use protocol *A* to establish a connection to network *B*, then they use protocol *B* to communicate with users on network *B*. Users on network *B* still use the same access protocol (*B*). This approach is transparent to users on one network but not to users on the other network.
3. *Protocol complementation.* This type of interworking is related to the layering of communicating protocols. Given protocol *A* of a layer in one network that has to interwork with protocol *B* of a layer in another network, a virtual layer can be added on top of *A* and *B* to provide a uniform view to users. Users on both networks need not be aware of the fact that there are different networks in the system. Gateways still need to be inserted between the networks. However, unlike gateways for protocol conversion, gateways for protocol complementation implement the uniform protocol of the virtual layer. Access protocols are changed for users in both networks. As a result, no transparency exists for users in this approach.

There has been very little work on the formal modeling of the last two approaches to protocol interworking. For the second approach, the substitution operation suggested in (Teng, 1980) seems to be promising. For the third approach, service specification will be a key issue. Much work has been done on protocol synthesis from service specification to protocol specification (see Section 6). The third approach seems to require just the reverse process, namely, how to generate a protocol for the new mutual layer from a given protocol and a service specification. How can the synthesis process be applied to this problem is a very challenging task.

9. Implementation and Conformance Testing

The final goal of a protocol design is successful incorporation of the protocol into an actual implementation. In particular, we are interested in *computer-automated implementation* of the protocol: the machine-readable formal specification could be translated or compiled directly into software or hardware for the final product. However, it is doubtful that the complete protocol specification can ever be translated or compiled directly into a

software or hardware implementation. Therefore, there is a need to test an implementation to determine if the implementation is indeed in conformance with the protocol specification; such a test is commonly called *conformance testing.*

Because of the complexity and difficulty of the problems associated with automatic implementation and conformance testing, there has been little progress made in this area of protocol engineering. In this section, we briefly describe some progress made in this area and suggest future research efforts needed.

9.1 Automated Implementation

In the past few years, there have been several experimental efforts in direct compilation of a protocol into parts of the code required for an implementation. Due to many hardware idiosyncracies, a substantial portion of an implementation must be hand-coded; but there is the hope that up to 60% of the necessary code can be automatically implemented.

For many years, IBM Systems Network Architecture (SNA) has been formally defined in terms of a meta-implementation language, called Format and Protocol Language (FAPL). The meta-implementation serves as a reference for actual implementations of the communication protocol it defines. Actual implementations must match the meta-implementation externally, but need not do so internally. Compilation of the protocol was carried out in two major steps (Nash, 1983). First the FAPL compiler was used to expand the FAPL specification into an intermediate language, PL/S, and the required manual code was also written in PL/S. Then the entire code was compiled and assembled into the appropriate machine code (in this case, the IBM 8100 Information System). The use of a semi-automated technique substantially reduced implementation time. Recently, Fleishmann *et al.* (1987) have proposed a technique using a new language, called PASS, to compile a protocol specification in the OSI session layer into Pascal.

Based on an EFSM model (see Section 3.3), the National Institute of Standards and Technology (NIST; formerly, National Bureau of Standards, NBS) has developed a language (actually a predecessor and subset of Estelle) to describe a subset of the OSI File Transfer Protocol. The formal specification was then compiled into the language C and about 40% of the code in C could be produced automatically (Linn, 1984; Mills, 1984). Other semi-automatic implementations based on Estelle include works by Serre *et al.* (1986) and by Blumer and Tenney (1982). Both works were concerned with a transport-level protocol and were able to automatically produce about half of the code required for the implementations.

A majority of protocol implementations, including those mentioned above,

are software programs for conventional uniprocessor architectures. Such programs have ranged from monolithic code to fairly complex software systems. Due to recent advances in VLSI technology, Krishnakumar *et al.* (1987) have proposed a systematic approach to the problem of protocol implementation in hardware from formal specifications. They proposed a method for generating VLSI layouts from formal protocol specifications, which are based on the CFSM model (see Section 3.1). Their method is based on a systematic partitioning of protocol functions in a hierarchical manner. This decomposition results in a flexible architecture that can implement many different protocols. They used the Link Access Protocol on the D-channel ISDN protocol, LAPD, as an example to illustrate their methodology. The major advantage of their approach lies in the area of design effort—reducing the implementation time from a few years to a few months.

9.2 Conformance Testing

The testing of a protocol implementation is the final phase in the development of a protocol design. In the context of the OSI Reference Model (see Fig. 2), particular attention is given to the methods by which protocol implementations can be tested for conformance with the protocol specifications. It is now widely accepted that OSI conformance testing is crucial to the achievement of the objective of OSI (Rayner, 1987).

A considerable amount of work has already been done in the area of testing OSI products for conformance to the standards. The major areas of research in protocol testing are in (1) test methods, (2) test suite design, and (3) test system implementation. Extensive efforts have been done in the United Kingdom by the National Physical Laboratory (Rayner, 1985), in West Germany at GMD (Burkhardt *et al.*, 1985), in France (Ansart, 1982), in Canada (Sarikaya and Bochmann, 1982, 1984) and in the United States at NIST (Linn and McCoy, 1983; Linn and Nightingale, 1983; Linn, 1984). All the work that has been done focuses on one of the three areas mentioned above. An approach to conformance testing has already reached the draft proposal stage in ISO (Rayner, 1987).

A *test suite* is defined to be a number of tests designed to verify the conformance of a protocol implementation to the protocol standard. A conformance test suite for a particular protocol tests all mandatory and optimal features of the protocol over the range of parameters and variations. In order to verify dynamic conformance requirements, live testing using a standard conformance test suite is performed.

Test methods are classified based on what outputs from the protocol entity under test are observed and what inputs to it can be controlled. A given method is described by identifying the points closest to the entity under test at

which control and observation are to be exercised. Three test methods have been proposed: (1) local test, (2) distributed test, and (3) remote test. These methods can be further classified according to the number of layers being tested: single-layer vs. multi-layer.

Figure 30 shows the general logical design of a test system currently in use by the National Physical Laboratory in the United Kingdom (Cowin *et al.*, 1983). The NIST system is similar and is discussed by Nightingale (1982). The local and distributed test methods require the use of an upper tester (UT). In Fig. 30, the UT is called the Test Responder (TR) and its purpose is to control and observe the primitives within the system in which the Implementation Under Test (IUT) being tested resides. The TR should be as simple as possible, and at the same time it should be flexible enough to be able to perform any test that is desired. The design of the TR depends on the environment it is going to

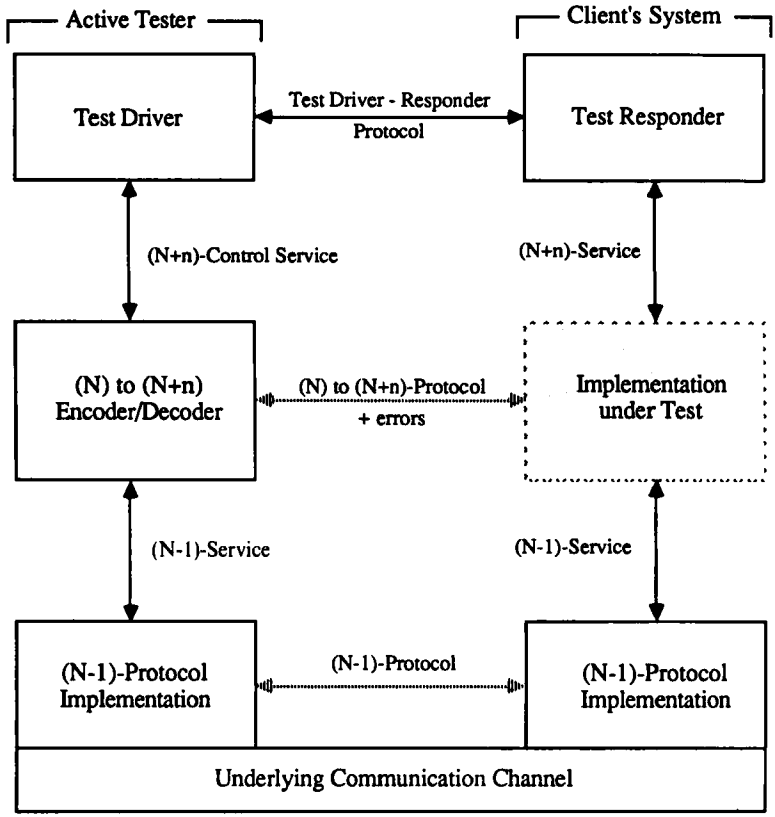


FIG. 30. General logical architecture

run in. A number of different designs for the TR have been proposed in the literature as follows:

1. The manual test responder (Palazzo *et al.*, 1983).
2. The scenario interpreter (Nightingale, 1982).
3. Finite state machines (Pavel and Dwyer, 1984).
4. Code interpreters (Burkhardt *et al.*, 1985).
5. Ferry concepts (Zheng and Rayner, 1985).

All of the test methods require a lower tester (LT), whose purpose is to control and observe the primitives within the system. In Fig. 30, the LT is called the Active Tester (AT) and consists of two components: the Test Driver (TD) and the Encoder/Decoder (E/D). The TD is the peer of the TR in the system under test; its major purpose is to control the operation of each test. The E/D is the peer of the IUT in the system under test; its major purpose is to encode and decode the message of the protocol in the IUT. Several different designs for the AT have been proposed in the literature as follows:

1. Reference implementation AT (Nightingale, 1982).
2. Reference implementation with error generator AT (Cowin *et al.*, 1983).
3. Protocol E/D AT (Cowin *et al.*, 1983).

There is a need to synchronize the activity of the UT and the LT. This can be accomplished by a Test Driver-Responder Protocol, as shown in Fig. 30. There are two main design choices, depending on where to operate the connection and how test events are related.

Even though many test systems have been implemented, more experience is needed to find out how well they can detect all kinds of errors. In the area of test design, work is needed to improve the way that the tests are generated. Some automatic test generation from the protocol specification is currently being done (Sabnani and Dahbura, 1983; Aho *et al.*, 1988), but a large percentage of the tests are still generated by hand. ISO is currently working to standardize OSI conformance testing (Rayner, 1987).

10. Automated Protocol Design

In recent years some progress has been made in creating an integrated set of tools for automated protocol design. The objective is to provide automated tools to lighten the task of the protocol designer while at the same time achieving a thorough analysis in the face of great complexity. These tools

provide assistance in the specification, validation, verifications, (partial) implementation, and conformance testing of protocols. Several realistic protocols have been analyzed and developed using such tools. In this section we present some of the automated systems that have been reported in the literature.

10.1 IBM System

Zafiropulo *et al.* (1980) at the IBM Zurich Research Laboratory have developed an interactive tool to facilitate protocol design. In their approach finite-state automata (FSA) are used as formal models. Two methods of analyzing protocol behavior are incorporated into the system, both of which can be used for either validation or synthesis.

The first method, the perturbation technique (West, 1978b; Zafiropulo, 1978a), has been used extensively to examine existing protocols, such as the X.21 (West and Zafiropulo, 1978) and the IBM token ring (Rudin, 1982). The second method, based on a set of production rules, has been incorporated into an automated synthesis system. Their initial attempt at protocol synthesis is one of the earliest in the field (Zafiropulo, 1978b; Zafiropulo *et al.*, 1979).

However, their tool does not provide any guidelines for helping the designer assign the entering state of each transition. If these states are not properly assigned, the resulting protocol may create deadlocks or livelocks; thus, the correctness is not guaranteed and further validation is required.

10.2 PROSPEC System

The PROSPEC system, developed at the University of Texas at Austin by Lam *et al.* (1986), also uses the model of communicating finite-state automata. It is constructed in a modular fashion, with each important function of the system being realized by a tool. The hierarchy of tools in the system is shown in Fig. 31. The protocol designer can invoke each tool independently to specify protocols graphically and also to verify protocols by looking at displays of reachability graphs. However, the graphical interface is not the most important element of PROSPEC. Its attractiveness lies in the designer's ability to access tools that implement techniques for managing the complexity of protocol specification and verification and for the modular construction of protocols.

The *resolution* of a protocol system has been proposed by Lam and Shankar (1984) as a basis for developing abstraction techniques to simplify the analysis and construction of multifunction protocols. They have developed the method of *projections* for constructing *image protocols*, each of which is specified just like any real protocol but is smaller than the original protocol. Obviously,

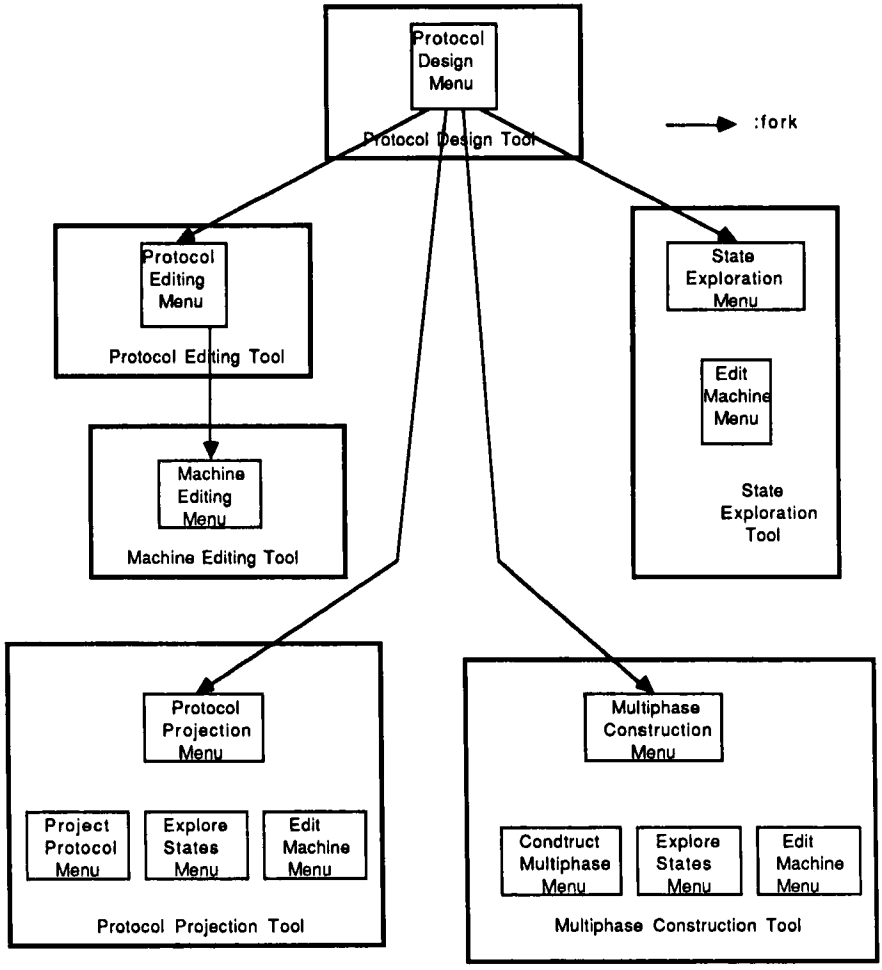


FIG. 31. Structure of the PROSPEC system (University of Texas at Austin)

fewer logical properties are observable and verifiable in an image protocol than in the original protocol. This approach was found to be very effective for the analysis of multifunction protocols that are not easily decomposable into different modules for implementing different functions, due to the use of shared variables and messages. A version of the HDLC protocol was verified using this method (Shankar and Lam, 1983).

The construction of a multifunction protocol from a *composition* of simple-function protocols is a much harder problem than the reverse problem

described above (i.e., the resolution problem). There is no easy method that corresponds to an *inverse projection* operation. However, Chow *et al.* (1984a) have observed that many realistic protocols go through different *phases* performing a distinct function in each phase. They presented a multiphase model for protocols and a methodology for constructing multiphase protocols (Chow *et al.*, 1985). They illustrated their methodology with the construction of several nontrivial multiphase protocols, including a version of the IBM BSC protocol for data link control (Chow *et al.*, 1985) and a high-level session control protocol (Chow *et al.*, 1984b).

PROSPEC has been developed on a SUN 2/120 workstation running 4.2 BSD UNIX. In addition to the graphical interface, the protocol designer can interactively access various tools that implement the method of projections, multiphase protocol constructions, and other features. The menu-selection facility relieves the designer of having to remember all the commands for interaction with PROSPEC.

10.3 Berkeley System

Ramamoorthy *et al.* (1985) at the University of California, Berkeley, have developed an automated protocol synthesizer (APS) that automatically generates the peer protocol entity from a single given local entity. The given entity is modeled by Petri nets, and if it satisfies certain prespecified constraints, the resulting protocols are guaranteed to possess desirable properties such as deadlock-freedom, boundedness, liveness, completeness, and proper termination. Their procedure consists of the following five steps (see Fig. 32):

1. Design a local entity model using Petri nets.
2. Translate the local entity model into its state-transition graph (STG1) by a state exploration procedure.
3. Check local properties of the given local entity model to make sure that it is well behaved. (This can be done by examining the structure of STG1).
4. Construct the peer state-transition graph (STG2) from STG1 according to certain well-designed transformation rules.
5. Construct the peer model in Petri nets from STG2.

Thus, the input of the APS is the Petri net specification of the giving entity, and the output will be the Petri net specification of its peer entity. Implemented on a VAX 11/780 machine using programming language C, the code size is about 3500 lines long and occupies 20K bytes of memory. It can accept a given entity model of up to 80 places and 150 transitions. It is a fairly

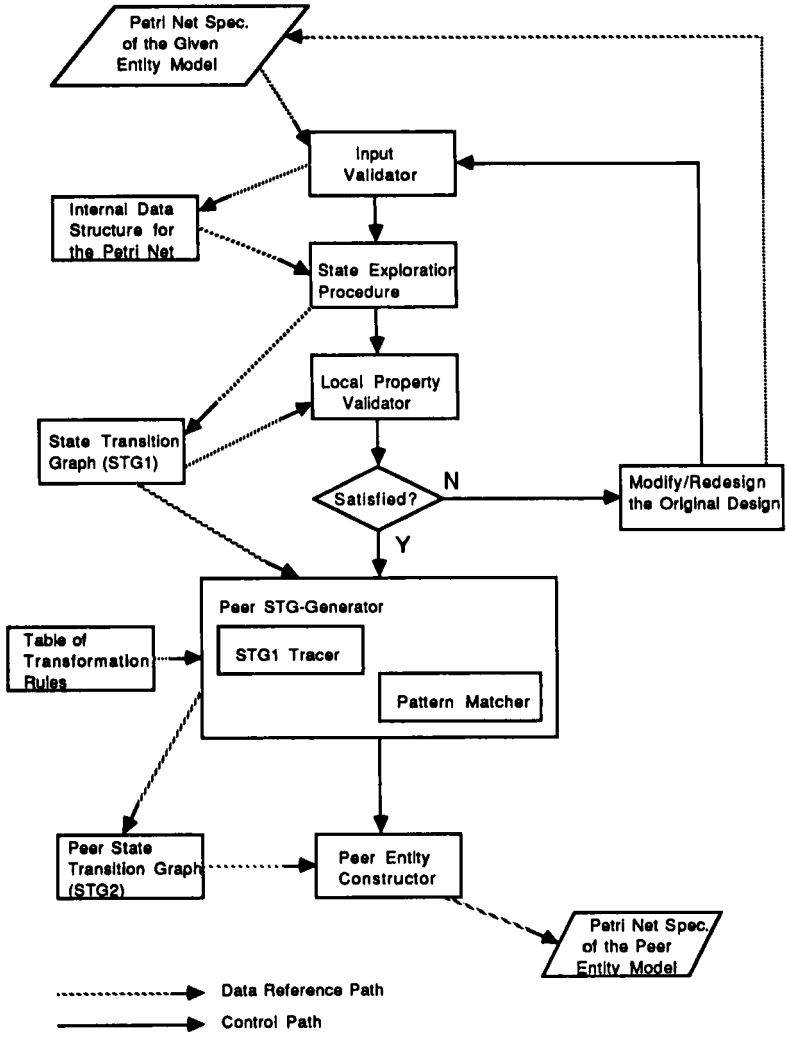


FIG. 32. Structure of the automated protocol synthesizer (University of California at Berkeley)

efficient computer-aided design tool and has been applied to a modified X.21 protocol to generate the peer entity model successfully. The X.21 protocol has 72 places and 122 transitions in the given entity, and the APS takes 3.70 seconds of CPU time to generate its peer entity model.

10.4 PANDORA System

The PANDORA system, an acronym for Protocol ANalysis, Design and OpeRation Assessment, aims to provide the protocol designer with a set of tools that can be used to design correct and efficient protocols (Holzmann, 1984). It consists of three major parts: analysis, synthesis, and real-time assessment (see Fig. 33).

In protocol analysis, the PANDORA system uses an algebraic model for protocol validation. The behavior of each communicating protocol entity is first modeled as a finite-state machine. The symbol sequences that can be accepted by these machines are then expressed in *protocol expressions* (Holzmann, 1982b), which are defined as *regular expressions* extended with

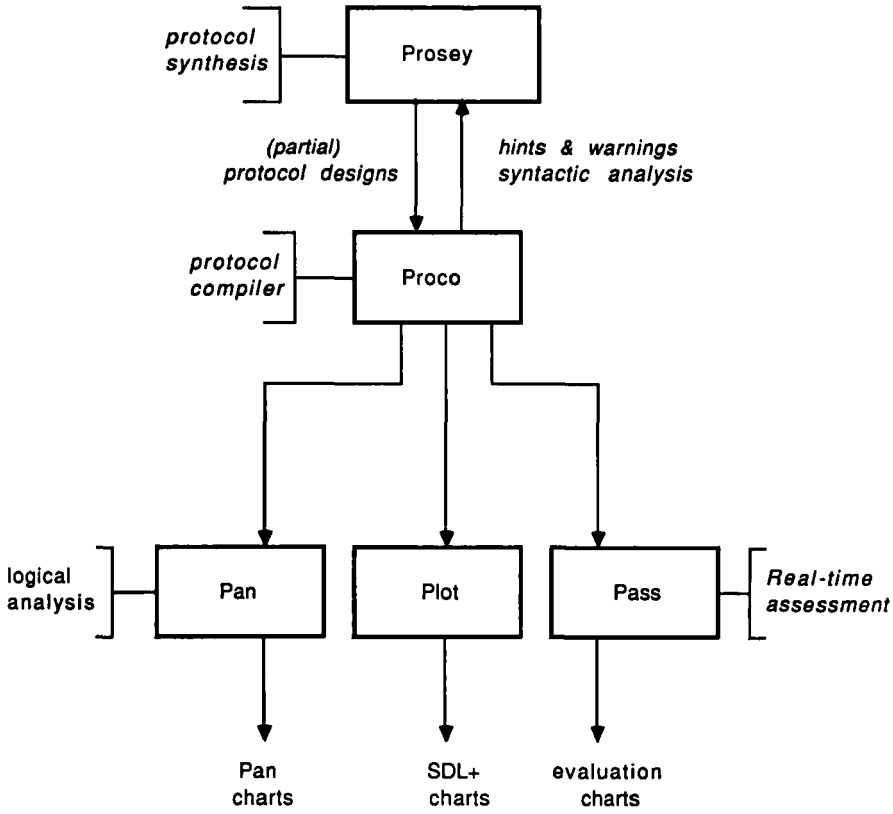


FIG. 33. Software structure of the PANDORA system (Delft University, Netherlands)

two new operators: division and multiplication. The interaction of the machines can be analyzed by combining protocol expressions via multiplication and algebraically manipulating the terms. Thus the problem of analyzing a protocol is transformed into one of analyzing an expression. Further, it is relatively easy to write a program that can accomplish this task efficiently for a fairly large class of protocols.

Compared to global state-space exploration techniques (West, 1978a), the validation method used in the PANDORA system allows for a number of important reductions in size and complexity of an analysis. These reductions are based on the notion of equivalent classes of execution sequences, and the validation process can now be restricted to examining just one characteristic sequence from each equivalent class. The gain over earlier reduction techniques (Rudin and West, 1982) is indeed significant.

The PANDORA system runs on two PDP 11/23 computers and its software is written in C and lives in a UNIX environment.

10.5 BBN/NIST System

Blumer and his associates have developed an automated technique for protocol development and its application to the specification, verification, and semi-automatic implementations of several realistic protocols (Blumer and Sidhu, 1983; Sidhu and Blumer, 1984). The major features of this technique are an augmented FSA model for protocol entities, specification of protocol entities in a Pascal-like language (Blumer and Tenney, 1982), a model used in building implementations from these specifications, and a collection of software tools. The software tools developed to support this technique provide the following services (see Fig. 34):

1. Syntax checking and type checking on specification.
2. Generation of FSM tables for a protocol entity, in various formats.
3. Compilation of a specification into a partial implementation.
4. Analyzing selected paths through protocol entity FSM.
5. Analyzing selected composite paths through several communicating protocol entity FSMs.
6. Verification of certain protocol properties.

In this system a protocol is first specified in a formalized protocol specification language. A specification compiler is then used to check the specification syntax and to generate code for a partial implementation. The compiler also generates FSM tables for the protocol, which are then used as input to the FSM analyzer for protocol analysis and verification. The analyzer

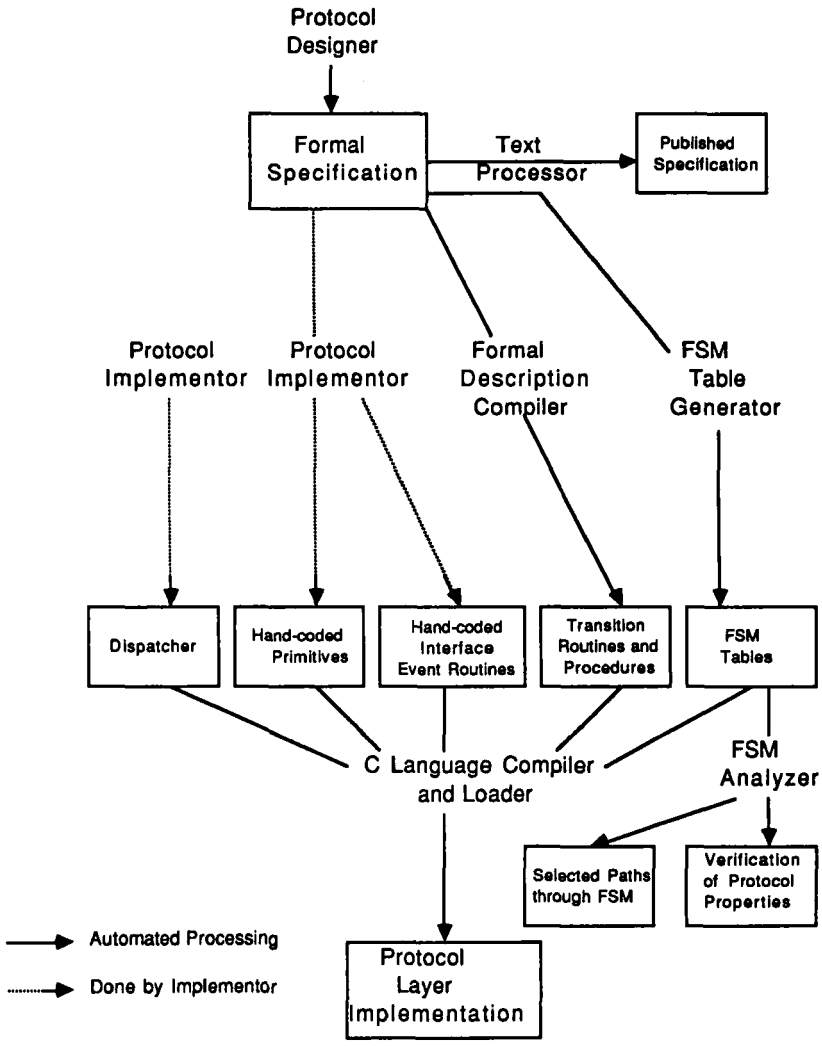


FIG. 34. Software tools for protocol development. (BBN/NIST)

analyzes possible protocol paths, and checks for certain protocol properties along each path. Information about each protocol path may be printed in several formats.

This technique has been used successfully in the development of several realistic protocols from NBST (TP4, TP2, Session and Message Protocols), DoD (TCP and IP), and IEEE 802.2 (LLC).

10.6 TTG/ETG Systems

Liu and his students at the Ohio State University have developed two automated validation systems for communication protocols. Called the TTG/ETG systems, both are based on a formal grammar model (the Transmission Grammar (Teng and Liu, 1978a, 1978b, 1980)). The TTG System (Lu, 1986) can handle timing constraints such as execution time of a protocol action, timeout intervals, timeout mechanisms, and transmission delay. In addition, it represents the communication medium in a different way

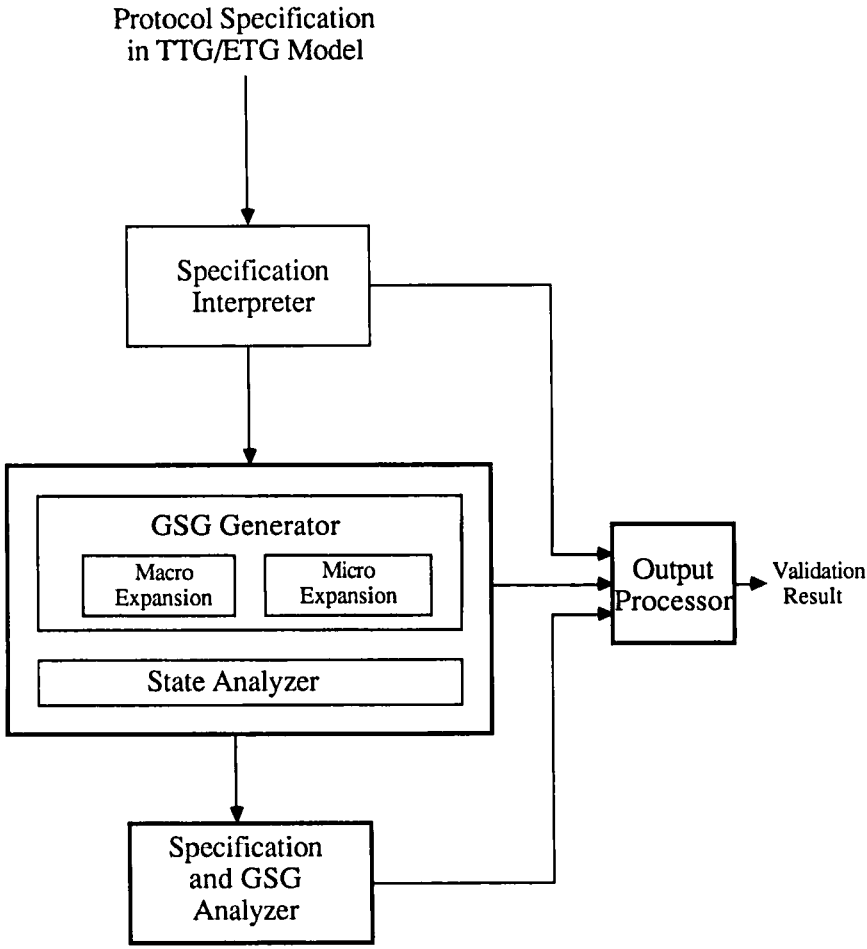


FIG. 35. Software structure of the TTG/ETG systems (The Ohio State University)

than other models, thereby allowing the protocol designer to model transmission errors, loss of messages, and out-of-sequence messages in a natural way. On the other hand the ETG System (Chu, 1989) can handle context variables such as sequence numbers.

The incorporation of timing information into the TTG model has an advantage of reducing validation efforts, since those global states that may not be generated under given timing constraints are excluded from the analysis. Moreover, the validation technique is based on a special kind of reachability analysis (a combination of Micro and Macro expansions, see Fig. 35) and can further reduce the global state space. Because of this expansion technique, the TTG system can validate not only more complex protocols, but asynchronous protocols as well as synchronous ones.

The TTG system has been developed on a VAX 11/780 machine, and its software is written in C and lives in a UNIX environment. It is portable to SUN workstations and has been used successfully to validate the ABP, the X.21, and the IBM token-ring protocol. The ETG system has been developed and runs under OSx on a Pyramid machine. It incorporates two global space reduction techniques and has been used to validate the ABP with a considerable amount of reduction in the total number of global states (Chu and Liu, 1989).

10.7 KBPV System

It is well known that conventional protocol validation based on reachability analysis suffers a great deal from the state explosion problem (see Section 4). Consequently, many variants of reachability analysis have been proposed in the literature to alleviate this problem. In Section 4.2, we have surveyed and evaluated these variant algorithms. One of the conclusions we reached is that none of the improved algorithms can totally supersede the others or even the conventional, exhaustive reachability analysis itself. In other words, each algorithm including the conventional one has advantages over the others under certain requirements and conditions. Thus we believe that a better protocol validation system should make these algorithms accessible to the protocol designer. This simply means to provide the protocol designer with a box of validation tools that implement various validation algorithms. We call this way of implementing the validation system the *tool box* idea.

Nevertheless, only providing the protocol designer with a tool box is not adequate unless the designer has the expertise of applying the right tool to the protocol of his or her concern. Unfortunately, such a requirement to the protocol designer is often too stringent to be realistic. First, the knowledge required to select a right algorithm or tool is dispersed in the literature and cannot be easily acquired by the designer. Secondly, the designer may be just a

novice user of the validation tools and may not be interested in understanding all the available validation algorithms.

Therefore, in addition to the tool box idea, we have proposed another idea called the *intelligent user-interface* to construct a user-friendly protocol validation system. The idea is to develop a knowledge-based interface that not only manages all the validation algorithms, but also acts as an intelligent assistant to help the protocol designer select and use these algorithms. It is natural to bring in the knowledge-based techniques here because the process of guiding a designer to select and use the most appropriate validation algorithm is basically *symbolic*.

The structure of such a *knowledge-based protocol validation* system (KBPV system) is illustrated in Fig. 36. Note that the symbolic (non-procedural) process of the system, namely the knowledge-based interface, is on the top of the algorithmic (procedural) processes implemented as a tool box of collection of validation algorithms. This kind of system is now getting attention from the AI community and is called the coupled system because both symbolic and algorithmic computing are coupled in the same system.

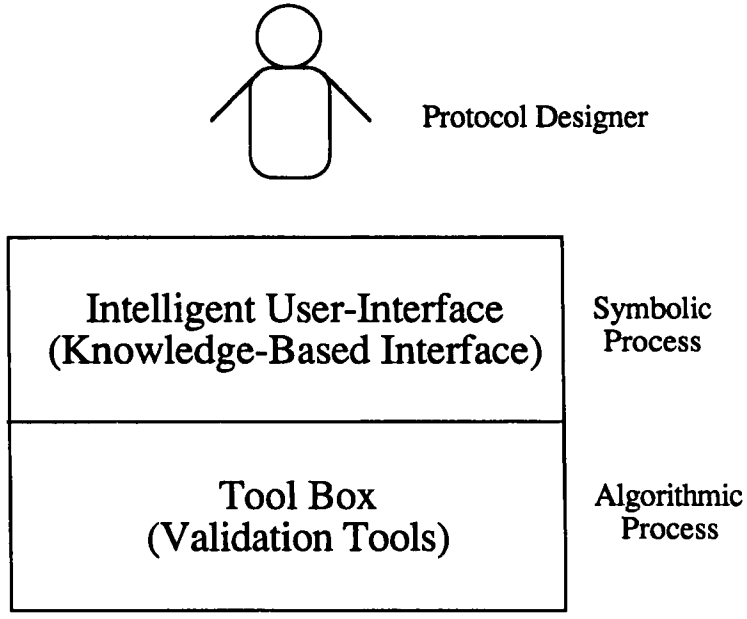


FIG. 36. Structure of the knowledge-based protocol validation system

In the first stage of our development, we have included the following six validation algorithms in the tool box:

1. Fair Progress Validation (Rudin and West, 1982; Gouda and Han, 1985).
2. Maximal Progress Validation (Gouda and Yu, 1984a).
3. Reduced Reachability Analysis (Itoh and Ichikawa, 1983).
4. Vuong's Reachability Analysis (Vuong and Cowan, 1982b).
5. Exhaustive Reachability Analysis (West, 1978a).
6. Protocol Validation Testing (Lin *et al.*, 1987).

Among the algorithms listed above, the fifth and sixth are supported by the PTG validation tool described in Section 4.4; the first four algorithms are supported by four separate tools recently developed. In fact, all these tools are developed by modifying an existing, conventional tool called TG (Lu, 1986). Every tool can accept protocol specifications in either Transmission Grammar (TG) or Probabilistic Transmission Grammar (PTG). Note that different tools may have different uses, and some of them may be quite complicated. Nevertheless, through the guidance and control of the intelligent user-interface, the protocol designer should have no difficulty in utilizing the full power of these tools. Our design of the intelligent user-interface is largely influenced by the idea behind the CSRL (Conceptual Structures Representation Language), a high-level language tuned specifically for implementing diagnostic expert systems (Bylander and Mittal, 1986). In CSRL, a specific organizational technique called *hierarchical classification* and a specific problem-solving strategy called *establish-refine* are employed to design a knowledge-based system. We believe that the structure and problem-solving strategy demonstrated by CSRL is quite suited in our domain of building an intelligent user-interface for the protocol validation system. The reasons are argued as follows:

1. The decision procedure of which algorithm to use in validating a protocol can be organized as a classification hierarchy of three levels as shown in Fig. 37.
2. The establish-refine control can be used as a search strategy in identifying the protocol under validation with an appropriate algorithm at the tip of the hierarchy.

To give more details, we briefly describe how this whole process works. From the root of the tree, the specialist (or concept) "protocol validation" first tries to establish itself. If successful, the succeeding refinement of it will pass

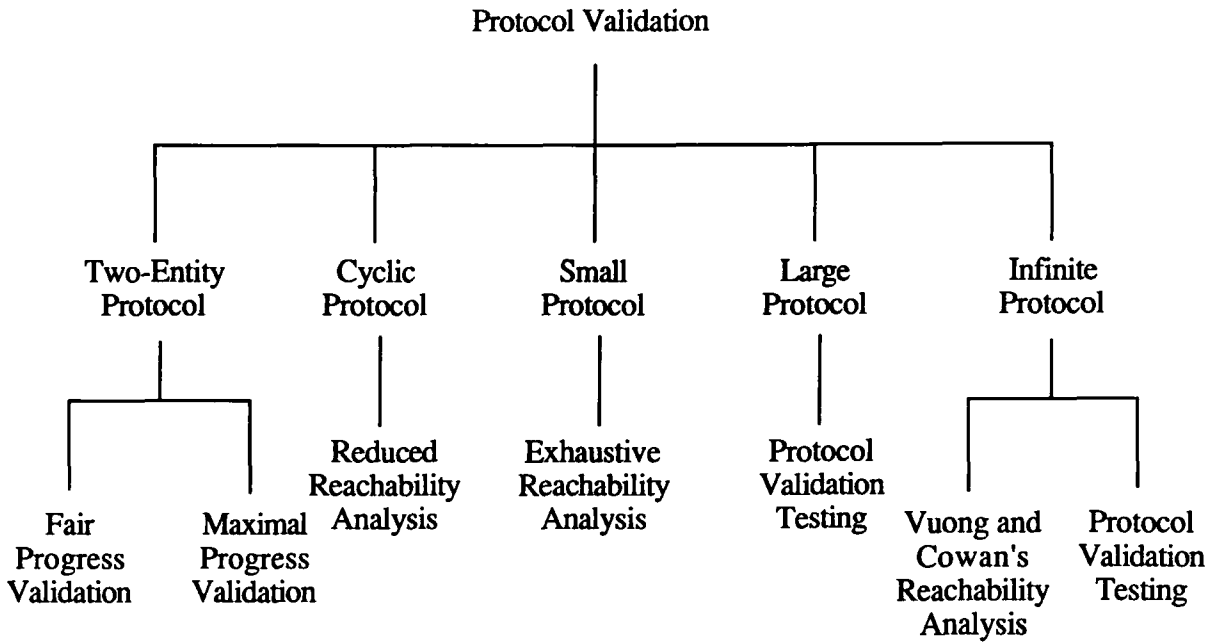


FIG. 37. The classification hierarchy of the intelligent user-interface

the control to the second level of specialists. The specialists in the second level then repeat the same process; they first try to establish themselves, and, if successful, may refine further down the tree after being granted by its superspecialist; otherwise, all its subspecialists will be excluded from further consideration. If a specialist at the tip of the hierarchy (the third level of the tree) establishes itself, it essentially means the feasibility of a specific validation algorithm to the protocol. By this process the validation algorithms suited for validating the protocol and their comparative scores can be determined. In our domain, the establishment or rejection of a concept is primarily based on (1) the formal protocol specification and (2) the interaction between the designer and the system during the establish-refine process. Note that knowledge rules are distributed to each specialist. We acquire those rules directly from the literature. The rules used by each specialist are given by Liu (1988).

A prototype of the KBPV system has been developed. It consists of approximately 10,000 lines of C code. A user manual for the KBPV System has been prepared (Liu, 1988).

11. Conclusion

The preceding sections have described various aspects of protocol engineering, a rapidly growing area of research in computer communications. A protocol engineering system allows the protocol designer to express the protocol formally, test its specification for correctness (validation and verification), obtain some early indication of how it would perform, compile major parts of the implementation directly from the formal specification, and finally, test the resultant implementation to assure that it conforms to the specification (implementation verification or conformance testing). These tasks are performed iteratively until a correct and efficient protocol is developed. The protocol engineering system can also be used by the protocol designer for protocol synthesis and protocol conversion.

As protocol design becomes more and more important due to the proliferation of computer communications, the need to use computer-aided design in the whole life cycle of protocol development becomes obvious. As described in Section 10, current protocol design systems do not provide enough support to help the designer make use of a variety of tools available to him or her. We believe that the incorporation of a knowledge-based system can help in those aspects, as they have already done so in other engineering disciplines (Sriram, 1986). A case in point is from the field of software engineering as reported in (Mostow, 1985; Simon, 1986), where knowledge-based systems are used to help automate the whole life cycle of software development. Since protocols are a special class of concurrent programs that

are communication-intensive, it is expected that those ideas and techniques developed for software engineering can be applied to protocol engineering as well.

A knowledge-based system is a new way of encoding human expertise into mechanically manipulable forms (Denning, 1986). It consists of two components: knowledge base and inference engine. The knowledge base, which corresponds to a program in conventional automatic problem-solving systems is a collection of encoded knowledge expressed in some formal representation. The inference engine, which corresponds to an interpreter in conventional systems, is a control mechanism to manipulate the representation in the knowledge base. These two components together provide a new regime of problem solving that deals with the encoding of the human's expertise much better than any standard procedure language.

The incorporation of knowledge-based systems into the protocol design process can be done in many ways. For example, program transformation techniques (Balzer, 1985; Fickas, 1985) can be used in deriving protocol specifications from given service specifications. Other AI techniques, such as search algorithms and theorem-proving can be used to reduce the global space search and to help correctness proving, respectively, in protocol validation and verification. Therefore, it is expected that both AI techniques and computer-aided software engineering (CASE) methodologies will play an important role in the future development of prootocol engineering.

ACKNOWLEDGMENTS

I wish to express my appreciation to many people who have been involved with the protocol engineering project. Special thanks are due to my former students Drs. Albert Y. Teng, L. David Umbaugh, H. A. Paul Lin, N. C. Liu, C. S. Lu, F. Joseph Lin, and P. Mark Chu. The research results reported in this article are mainly based on their dissertation research under my supervision. Thanks are also due to Dr. L. Chiu, Dr. Ian Y. Chiou, Dr. H. Yoon, W. S. Chen, I. E. Liao, C. C. Wu and Y. W. Yao, who have contributed directly or indirectly, Other students currently involved in the project are J. Chang, Y. I. Chang, C. M. Dorcy Huang, H. S. Jang, J. C. Shu, H. W. Jeng, and S. H. Sarah Yu.

Research reported herein has been supported by a series of contracts from U. S. Army Communications-Electronics Command (CECOM), Fort Monmouth, New Jersey. My students and I are very grateful to Dr. Charles J. Graff, the project monitor, for his continuing support and for his foresight in sponsoring the research project on protocol engineering.

REFERENCES

Aggarwal, S., and Sabnani, K. (1986). Formal specification of a file transfer protocol. *Proc. IEEE INFOCOM*, pp. 47–57.

Aggarwal, S., Kurshan, R. P., and Sabnani, K. (1983). A calculus for protocol specification and validation. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 3rd*, pp. 19–34.

Aho, A. V., *et al.* (1988). An optimization technique for protocol conformance test generation based on UIO sequences and Rural Chinese postman tours. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 8th*, pp. 75–76.

Amer, P. D., Ceceli, F., and Juanole, G. (1988). Formal specification of ISO virtual terminal in Estelle. *Proc. IEEE INFOCOM*, pp. 623–630.

Amer, P. D., Pridor, A., and Schmidt, J. (1988). Expansion of transitions in Estelle formal specifications. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 8th*, pp. 159–170.

Anderson, D. P., and Landweber, L. H. (1984a). Protocol specification by real time attribute grammars. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 4th*, pp. 457–466.

Anderson, D. P., and Landweber, L. H. (1984b). A grammar-based methodology for protocol specification and implementation. *Proc. ACM/IEEE Data Comm. Symp., 9th*, pp. 63–70.

Ansart, J. P. (1982). GENEPI/A—a protocol independent system for testing protocol implementation. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 2nd*, pp. 523–528.

Ansart, J. P. (1985). Issues and tools for protocol specification. *In* "Distributed Systems: Methods and Tools for Specification" (W. W. Alford, *et al.*, eds.), pp. 481–538. Springer-Verlag, Berlin, W. Germany.

Ansart, J. P., Rafiq, O., and Chari, V. (1982). PDIL—protocol description and implementation language. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 2nd*, pp. 101–112.

Apt, K. R., Francez, N., and De Roever, W. P. (1980). A proof system for communicating sequential processes. *ACM TOPLAS* **2** (3), 359–385.

Balzer, R. (1985). A 15 year perspective on automatic programming. *IEEE Trans. Software Engineering* **SE-11** (11), 1257–1267.

Barbeau, M., and Sarikaya, B. (1988). A computer-aided design tool for protocol testing. *Proc. IEEE INFOCOM*, pp. 86–95.

Bartlett, K. A., Scantlebury, R. A., and Wilkinson, P. T. (1969). A note on reliable full-duplex transmission over half-duplex lines. *Comm. ACM* **12** (5), 260–261.

Berthomieu, B., and Menasche, M. (1983). An enumerative approach for analyzing time Petri nets. *Proc. IFIP Information Processing 83*, pp. 41–46.

Billington, J., Wilbur-Ham, M. C., and Bearman, M. Y. (1985). Automated protocol vertification. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 5th*, pp. 59–70.

Blumer, T. P., and Sidhu, D. P. (1983). Experience with an automated protocol development system. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 3rd*, pp. 369–380.

Blumer, T. P., and Sidhu, D. P. (1986). Mechanical verification and automatic implementation of communication protocols. *IEEE Trans. Software Engineering* **SE-12** (8), 827–842.

Blumer, T. P., and Tenney, R. L. (1982). A formal specification technique and implementation method for protocols. *Computer Networks* **6** (3), 201–217.

Bochmann, G. V. (1975). Logical verification and implementation of protocols. *Proc. ACM/IEEE Data Comm. Symp., 4th*, pp. 8.5–8.20.

Bochmann, G. V. (1978). Finite state description of communication protocols. *Computer Networks* **2**, 362–372.

Bochmann, G. V. (1980). A general transition model for protocols and communications services. *IEEE Trans. Comm.* **COM-28** (4), 643–650.

Bochmann, G. V. (1981). The use of formal description techniques for OSI protocols. *Proc. Nat. Telecomm. Conf.*, pp. F8.6.1–F8.6.6.

Bochmann, G. V., and Gecsei, J. (1977). A unified method for the specifications and verfication of protocols. *Proc. IFIP Information Processing 77*, pp. 229–234.

Bochmann, G. V., and Gotzhein, R. (1986). Deriving protocol specifications from service specifications. *Proc. ACM SIGCOMM '86 Symp.*, pp. 148–156.

Bochmann, G. V., and Sunshine, C. A. (1980). Formal methods in communication protocol design. *IEEE Trans. Comm.* **COM-28** (4), 624–631.

Bochmann, G. V. *et al.* (1982a). Some experience with the use of formal specifications. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 2nd*, pp. 171–186.

Bochmann, G. V. *et al.* (1982b). Experience with formal specifications using an extended state transitions model. *IEEE Trans. Comm.* **COM-30** (12), 2506–2513.

Bolognesi, T., and Brinksma, E. (1987). Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems* **14** (1), 25–29.

Bolognesi, T., and Rudin, T. (1984). On the analysis of time-dependent protocols by network flow algorithms. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 4th*, pp. 491–514.

Brand, D., and Zafiropulo, P. (1980). Synthesis of protocols for an unlimited number of processes. *Proc. NBS Trends and Applications Conf.*, pp. 29–40.

Brand, D., and Zafiropulo, P. (1983). On communicating finite-state machines *J. ACM* **30** (2), 323–342.

Brinksma, E., and Karjoth, G. (1984). A specification of the OSI transport service in LOTOS. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 4th*, pp. 227–252.

Budkowski, S., and Dembinski, P. (1987). An introduction to Estelle: a specification language for distributed systems. *Computer Networks and ISDN Systems* **14** (1), 3–23.

Burkhardt, H. J., Eckert, H., and Giessler, A. (1985). Testing of protocol implementation: a systematic approach to derivation of test sequences from global protocol specifications. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 5th*, pp. 461–482.

Bylander, T., and Mittal, S. (1986). CSRL: a language for classificatory problem solving and uncertainty handling. *IEEE AI Magazine* **7** (3), 66–77.

Calvert, K. L., and Lam, S. S. (1987). An exercise in deriving a protocol conversion. *Proc. ACM SIGCOMM '87 Workshop*, pp. 151–160.

Castanet, R., Dupeux, A., and Guitton, P. (1985). ADA, a well suited language for specification and implementation of protocols. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 5th*, pp. 247–258.

Choi, T. Y. (1983). A structured approach to the analysis and design of finite state protocols. Ph.D. dissertation, Georgia Institute of Technology, Atlanta, Georgia.

Choi, T. Y. (1986). A sequence method for protocol construction. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 6th*, pp. 307–321.

Choi, T. Y., and Miller, R. E. (1983). A decomposition method for the analysis and design of finite state protocols. *Proc. ACM/IEEE Data Comm. Symp., 8th*, pp. 167–176.

Chow, C. H. (1985). A discipline for the verification and modular construction of communication protocols. Ph.D. dissertation, University of Texas, Austin, Texas.

Chow, C. H., Gouda, M. G., and Lam, S. S. (1984a). On constructing multi-phase communication protocols. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 4th*, pp. 57–68.

Chow, C. H., Gouda, M. G., and Lam, S. S. (1984b). An exercise in constructing multi-phase communication protocols. *Proc. ACM SIGCOMM '84 Symp.*, pp. 42–44.

Chow, C. H., Gouda, M. G., and Lam, S. S. (1985). A discipline for constructing multiphase communication protocols. *ACM Trans. Computer Systems* **3** (40), 315–343.

Chu, P. M. (1989). Towards automating protocol synthesis and analysis. Ph.D. dissertation, Ohio State University, Columbus, Ohio.

Chu, P. M., and Liu, M. T. (1988a). Synthesizing protocol specifications from service specifications in FSM models. *Proc. IEEE Computer Networking Symp.*, pp. 173–182.

Chu, P. M., and Liu, M. T. (1988b). Protocol synthesis in a state-transition model. *Proc. IEEE COMPSAC*, pp. 505–512.

Chu, P. M., and Liu, M. T. (1989). Global state graph reduction techniques for protocol validation in the EFMS model. *Proc. IEEE Phoenix Conf. on Computers and Comm., 8th*, pp. 371–377.

Chung, R. S. Y. (1984). A methodology for protocol design and specification based on an extended state transition model. *Proc. ACM SIGCOMM '84 Symp.*, pp. 34–41.

Cowin, G. W., Hale, R. W. S., and Rayner, D. (1983). Protocol product testing—some comparisons and lessons *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 3rd*, pp. 477–493.

Dahbura, A., and Sabnani, K. (1988). An experience in estimating fault coverage of a protocol test. *Proc. IEEE INFOCOM*, pp. 71–79.

Danthine, A. (1980). Protocol representation with finite state models. *IEEE Trans. Comm.* **COM-28** (4), 632–643.

Day, J. D., and Zimmerman, H. (1983). The OSI reference model. *Proc. IEEE* **71** (12), 1334–1340.

Denning, P. J. (1986). Towards a science of expert systems. *IEEE Expert Magazine* **1** (2), 80–85.

Diaz, M. (1982). Modelling and analysis of communication and cooperation protocols using Petri net based models. *Computer Networks* **6** (6), 419–441.

Diaz, M. Ansart, J. P., Azema, P., and Chari, V., eds. (1989). "The Formal Description Technique Estelle." North-Holland, Amsterdam, The Netherlands.

Dickson, G. J., and de Chazal, P. (1983). Status of CCITT description techniques and application to protocol specification. *Proc. IEEE* **71** (12), 1346–1355.

DiVito, B. L. (1982). Verification of communication protocols and abstract process models. Ph.D. dissertation, University of Texas, Austin, Texas.

Dong, S. T. (1983). The modeling, analysis and synthesis of communications protocols. Ph.D. dissertation, University of California, Berkeley, California.

Duc, N. Q., and Chew, E. K. (1986). ISDN protocol architecture. *IEEE Comm. Magazine* **23** (3), 15–22.

Engelbrecht, J. R., Kritzinger, P. S., and Rudin, H. (1985). Predicting protocol performance from a meta-implementation *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 5th*, pp. 349–362.

Fickas, S. F. (1985). Automating the transformational development of software. *IEEE Trans. Software Engineering* **SE-11** (11), 1268–1277.

Fleischmann, A., Chin, S. T., and Effelsberg, W. (1987). Specification and implementation of an ISO layer. *IBM Sys. Jour.* **26** (3), 255–275.

Floyd, R. W. (1967). Assigning meanings to programs. *Proc. Symp. in Applied Math.* **19**, 19–32.

Garg, K. (1985). An approach to performance specification of communication protocols using timed Petri nets. *IEEE Trans. Software Engineering* **SE-11** (10), 1216–1225.

Gerhart, S., *et al.* (1980). An overview of AFFIRM: a specification and verification system. *Proc. IFIP Information Processing 80*, pp. 343–348.

Good, D. J., and Cohen, R. M. (1979). Principles of proving concurrent programs in Gypsy. *Proc. ACM Symp. on Principles of Programming Languages, 6th*, pp. 42–54.

Gouda, M. G. (1984). Closed covers: to verify progress for communicating finite state machines. *IEEE Trans. Software Engineering* **SE-10** (6), 846–855.

Gouda, M. G., and Chang, C. K. (1984). A technique for proving liveness of communicating finite state machines with examples. *Proc. ACM Symp. on Principles of Distributed Computing, 3rd*, pp. 38–49.

Gouda, M. G., and Han, J. Y. (1985). Protocol validation by fair progress state exploration. *Computer Networks & ISDN Systems* **9**, 353–361.

Gouda, M. G., and The, K. S. (1985). Modeling physical layer protocols using communicating finite state machines. *Proc. ACM/IEEE Data Comm. Symp., 9th*, pp. 54–62.

Gouda, M. G., and Yu, Y. T. (1984a). Protocol validation by maximal progress state exploration. *IEEE Trans. Comm.* **COM-32** (1), 94–97.

Gouda, M. G., and Yu, Y. T. (1984b). Synthesis of communicating finite-state machines with guaranteed progress. *IEEE Trans. Comm.* **COM-32** (7), 779–788.

Gouda, M. G., Chow, C. H., and Lam, S. S. (1984). On the decidability of livelock detection in networks of communicating finite state machines. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 4th*, pp. 47–56.

Green, P. E. (1986). Protocol conversion. *IEEE Trans. Comm.* **COM-34** (3), 257–268.

Guttag, J. V. (1975). The specification and application to programming of abstract data types. Ph.D. dissertation, University of Toronto, Toronto, Canada.

Hailpern, B. T. (1980). Verifying concurrent processes using temporal logic. Ph.D. dissertation, Stanford University, Stanford, California.

Hailpern, B. T., and Owicki, S. (1980). Verifying network protocols using temporal logic. *Proc. NBS Trends and Applications Conf.*, pp. 18–28.

Hailpern, B. T., and Owicki, S. (1983). Modular verification of computer communication protocols. *IEEE Trans. Comm.* **COM-31** (1), 56–68.

Harangozo, J. (1977). An approach to describing a link level protocol with a formal language. *Proc. ACM/IEEE Data Comm. Symp., 5th*, pp. 4.37–4.49.

Hoare, C. A. R. (1969). An axiomatic basis for computer programming. *Comm. ACM* **12** (10), 567–583.

Hoare, C. A. R. (1978). Communicating sequential processes. *Comm. ACM* **21** (8), 666–677.

Holliday, M. A., and Vernon, M. K. (1987). A generalized timed Petri net model for performance analysis. *IEEE Transs. Software Engineering SE-13* (12), 1297–1310.

Holzmann, G. J. (1982a). Algebraic validation methods—a comparison of three techniques. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 2nd*, pp. 383–390.

Holzmann, G. J. (1982b). A theory for protocol validation. *IEEE Trans. Computers* **C-31** (8), 730–738.

Holzmann, G. J. (1984). The Pandora system: an interactive system for the design of data communication protocols. *Computer Networks* **8** (2), 71–79.

Holzmann, G. J. (1985). Tracing protocols. *AT&T Tech. Jour.* **64** (10), 2413–2433.

Holzmann, G. J. (1987). Automatic protocol validation in Argos: assertion proving and scatter searching. *IEEE Trans. Software Engineering* **SE-13** (6), 683–696.

Itoh, M., and Ichikawa, H. (1983). Protocol verification algorithm using reduced reachability analysis. *Trans of IECE of Japan* **E66** (2), 88–93.

Jain, P., and Lam, S. S. (1987). Modeling and verification of real-time protocols for broadcast networks. *IEEE Trans. Software Engineering* **SE-13** (8), 924–937.

Jurgensen, W., and Vuong, S. T. (1984). Formal specification and validation of ISO transport protocol components using Petri nets. *Proc. ACM SIGCOMM '84 Symp.*, pp. 75–82.

Kakuda, Y., and Wakahara, Y. (1987). Component-based synthesis of protocols for unlimited number of processes. *Proc. IEEE COMPSAC*, pp. 721–730.

Kakuda, Y., Wakahara, Y., and Norigoe, M. (1986). A new algorithm for fast protocol validation. *Proc. IEEE COMPSAC*, pp. 228–236.

Keller, R. M. (1976). Formal verification of parallel programs. *Comm. ACM* **19** (7), 371–384.

Krishnakumar, A. S., Krishnamurthy, B., and Sabnani, K. (1987). Translation of formal protocol specifications to VLSI devices. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 7th*, pp. 375–390.

Kritzinger, P. S. (1984). Analyzing the time efficiency of a communication protocol. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 4th*, pp. 527–540.

Krogdahl, S. (1978). Verification of a class of link-level protocols. *BIT* **18**, 436–448.

Krumm, H., and Drobnik, O. (1983). Specification, implementation and verification of communication services on the basis of CIL. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 4th*, pp. 301–316.

Krumm, H., and Drobnik, O. (1984). Transformation of constructive specifications of services and protocols into the logical language of CIL. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 4th*, pp. 31–46.

Lam, S. S. (1986). Protocol conversion—correctness problems. *Proc. ACM SIGCOM '86 Symp.*, pp. 19–29.

Lam, S. S. (1988). Protocol conversion. *IEEE Trans. Software Engineering* **SE-14** (3), 353–362.

Lam, S. S., and Shankar, A. U. (1982). Verification of communication protocols via protocol projections. *Proc. IEEE INFOCOM*, pp. 229–240.

Lam, S. S., and Shankar, A. U. (1984). Protocol verification via projections. *IEEE Trans. Software Engineering* **SE-10** (4), 325–342.

Lam, S. S., *et al.* (1986). Interactive verification and construction of communication protocols in PROSPEC, *Proc. IEEE INFOCOM*, pp. 67–75.

Lamport, L. (1983). Specifying concurrent program modules. *ACM TOPLAS* **5** (2), 190–222.

Lee, T. T., and Lai, M. Y. (1988). A relational algebraic approach to protocol verification. *IEEE Trans, Software Engineering* **SE-14** (2), 184–193.

Levin, G. M., and Gries, D. (1981). A proof technique for communicating sequential processes. *Acta Info.* **15**, 281–302.

Liao, I. E., and Liu, M. T. (1989). Incremental protocol verification using deductive database systems. *Proc. IEEE Int. Conf. on Data Engineering, 5th*, pp. 216–223.

Lin, F. J. (1988). An integrated approach to verification and performance analysis of communication protocols. Ph.D. dissertation, Ohio State University, Columbus, Ohio.

Lin, F. J., and Liu, M. T. (1988a). A formal model for protocol interworking in ISDN. *Proc. IEEE Int. Conf. Comm.*, pp. 107–113.

Lin, F. J., and Liu, M. T. (1988b). An integrated approach to verification and performance of communication protocols. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 8th*, pp. 125–140.

Lin, F. J., Chu, P. M., and Liu, M. T. (1987). Protocol verification using reachability analysis: the state explosion problem and relief strategies. *Proc. ACM SIGCOMM '87 Workshop*, pp. 126–135.

Lin, F. J., Liu, M. T., and Graff, C. J. (1989). On the verification of time-dependent protocol using timed reachability analysis. *Proc. Hawaii Int. Conf. on Sys. Sci., 22nd*, **z**, pp. 285–294.

Lin, H. A. (1983). A methodology for designing reliable communication protocols. Ph.D. dissertation, Ohio State University, Columbus, Ohio.

Lin, H. A., and Liu, M. T. (1982). Priority driven communication protocol design. *Proc. IEEE Int. Conf. on Distributed Computing Systems, 3rd*, pp. 371–378.

Lin, H. A., and Liu, M. T. (1986). Verification of CSP programs based on communication assertions. *Proc. IEEE Phoenix Conf. on Computers and Comm., 5th*, pp. 412–417.

Lin, H. A., Liu, M. T., and Graff, C. J. (1983a). A methodology for reliable communications protocol design. *Proc. IEEE Int. Conf. Comm.*, pp. 1338–1343.

Lin, H. A., Liu, M. T., and Graff, C. J. (1983b). Verification of a methodology for designing reliable communication protocols. *Proc. ACM/IEEE Data Comm. Symp., 8th*, pp. 141–149.

Linn, R. J. (1984). An evaluation of the ICST test architecture after testing class 4 transport. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 4th*, pp. 611–622.

Linn, R. J. (1985). The features and facilities of ESTELLE: a formal description technique based upon an extended finite state machine model. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 5th*, pp. 271–296.

Linn, R. J., and Favreau, J. P. (1988). Application of formal description techniques to the specification of distributed test systems. *Proc. IEEE INFOCOM*, pp. 96–109.

Linn, R. J., and McCoy, W. H. (1983). Producing tests for implementations of OSI protocols. *Proc. Int. Workshop on Protocol Specification, Testing, and Verification, 3rd*, pp. 505–520.

Linn, R. J., and Nightingale, J. S. (1983). Some experience with testing tools for OSI protocol implementations. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 3rd*, pp. 521–530.

Liu, M. T. (1978). Distributed loop computer networks. *In* "Advances in Computers" (M. C. Yovits, ed.), Vol. 17, pp. 163–221. Academic Press, New York.

Liu, M. T. (1988). Computer communication protocols for advanced communication systems. Technical Report No. CECOM-TR-87-K-A005-3, Ohio State University, Columbus, Ohio.

Liu, N. C. (1986). A methodology for specifying and analyzing communication protocols and services. Ph.D. dissertation, Ohio State University, Columbus, Ohio.

Liu, N. C., and Liu, M. T. (1984). CSP-based specification for network protocols and services. *Proc. IEEE Computer Networking Symp.*, pp. 95–102.

Liu, N. C., and Liu, M. T. (1986). Conformity analysis for communication protocols. *Proc. ACM SIGCOMM '86*, pp. 216–226.

Lu, C. S. (1986). Automated validation of communication protocols. Ph.D. dissertation, Ohio State University, Columbus, Ohio.

Lynch, W. C. (1968). Reliable full-duplex transmission over half-duplex telephone lines. *Comm. ACM* **11** (6), 362–372.

Marsan, M. A., Conte, G., and Balbo, G. (1984). A class of generalized stochastic Petri nets for performance evaluation of multiprocessor systems. *ACM Trans. Computer Systems* **2** (2), 93–122.

Menasche, M. (1985). PAREDE: an automated tool for the analysis of time(d) Petri nets. *Proc. IEEE Int. Workshop on Time Petri Nets*, 162–169.

Menasche, M., and Berthomieu, B. (1983). Time Petri nets for analyzing and verifying time dependent communication protocols. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 3rd*, pp. 161–172.

Merlin, P. M. (1976). A methodology for the design and implementation of communication protocols. *IEEE Trans. Comm.* **COM-24** (6), 614–621.

Merlin, P. M., and Bochmann, G. V. (1983). On the construction of submodule specifications and communication protocols. *ACM TOPLAS* **5** (1), 1–25.

Merlin, P. M., and Farber, D. J. (1976). Recoverability of communication protocols—implications of a theoretical study. *IEEEE Trans. Comm.* **COM-24** (9), 1036–1043.

Mills, K. L. (1984). Testing OSI protocols: NBS advances the state of the art. *Data Comm.* **13** (3), 277–285.

Milner, R. (1980). "A Calculus of Communicating System." Springer-Verlag, Berlin, W. Germany.

Molloy, M. K. (1982). Performance analysis using stochastic Petri nets. *IEEE Trans. Computers* **C-31** (9), 913–917.

Mostow, J. (1985). What is AI? and what does it have to do with software engineering? *IEEE Trans. Software Engineering* **SE-11** (11), 1253–1255.

Nash, S. C. (1983). Automated implementation of SNA communication protocols. *Proc. IEEE Int. Conf. Comm.*, pp. 1316–1322.

Nash, S. C. (1987). Formal and protocol language (FAPL). *Computer Networks and ISDN Systems* **14** (1), 61–77.

Nightingale, J. S. (1982). Protocol testing using a reference implementation. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 2nd*, pp. 513–522.

Nounou, N. (1986). Specification-based performance analysis of protocols. Ph.D. dissertation, Columbia University, New York.

Nounou, N., and Yemini, Y. (1984). Algebraic specification-based performance analysis of communication protocols. *Proc. Int. Workshop on Protocol Specification, Testing, and Verification, 4th*, pp. 541–560.

Nounou, N., and Yemini, Y. (1986). Development tools for communication protocols. *In* "Current Advances in Distributed Computing and Communications" (Y. Yemini, ed.), pp. 257–304. Computer Science Press, Rockville, Maryland.

Okumura, K. (1986). A formal protocol conversion method. *Proc. ACM SIGCOMM '86 Symp.*, pp. 30–38.

Palazzo, S., Fogliata, P., and Le Moli, G. (1983). A layer-independent architecture for a testing system of protocol implementations. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 3rd*, pp. 393–406.

Partsch, H., and Steinbruggen, R. (1983). Program transformation systems. *ACM Computing Survey* **15** (3), 199–236.

Pavel, J. R., and Dwyer, D. J. (1984). Some experiences of testing protocol implementations. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 4th*, pp. 657–680.

Peral, J. (1984). "Heuristics—Intelligent Search Strategies for Computer Problem Solving." Addison-Wesley, Reading, Massachusetts.

Piatkowski, T. F. (1981). An engineering discipline for distributed protocol systems. *Proc. IFIP Workshop on Protocol Testing—Towards Proof?*, pp. 177–215.

Piatkowski, T. F. (1983). Protocol engineering. *Proc. IEEE Int. Conf. Comm.*, pp. 1328–1332.

Piatkowski, T. F. (1986). The state of the art in protocol engineering. *Proc. ACM SIGCOM '86 Symp.*, pp. 13–18.

Pnueli, A. (1977). The temporal logic of programs. *Proc. IEEE/ACM Symp. on Foundations of Computer Science, 18th*, pp. 46–57.

Postel, J., and Farber, D. J. (1976). Graph modeling of computer communications protocols. *Proc. Texas Conf. on Computing Systems, 5th*, pp. 66–77.

Pozefsky, D. P., and Smith, F. D. (1982). A meta-implementation for system network architecture. *IEEE Trans. Comm.* **COM-30** (6), 1348–1355.

Prinoth, R. (1982). An algorithm to construct distributed systems from state-machines. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 2nd*, pp. 261–282.

Purushothaman, S., and Subrahmanyam, P. A. (1987). Reasoning about probabilistic behavior in concurrent systems. *IEEE Trans. Software Engineering* **SE-13** (6), 740–745.

Ramamoorthy, C. V., and Dong, S. T. (1982). Communication protocol synthesis. *Proc. IEEE COMPSAC*, pp. 217–225.

Ramamoorthy, C. V., Dong, S. T., and Usuda, Y. (1985). An implementation of an automated protocol synthesizer (APS) and its application to the X.-21 protocol. *IEEE Trans. Software Engineering* **SE-11** (9), 886–908.

Rayner, D. (1985). Towards standardized OSI conformance tests. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 5th*, pp. 441–460.

Rayner, D. (1987). OSI conformance testing. *Computer Networks and ISDN Systems* **14** (1), 79–98.

Razouk, R. R. (1981). Computer-aided design and evaluation of digital computer systems. Ph.D. dissertation, University of California, Los Angeles, California.

Razouk, R. R. (1982). Modeling X.25 using the graph model of behavior. *Proc. Int. Workshop on Protocol Specification, Testing, and Verification, 2nd*, pp. 197–214.

Razouk, R. R., and Estrin, G. (1980). Modeling and verification of communication protocols in SARA: the X.21 interface. *IEEE Trans. Computers* **C-29** (12), 1038–1052.

Razouk, R. R., and Phelps, C. V. (1984). Performance analysis using timed Petri nets. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 4th*, pp. 561–576.

Reed, G. M., and Roscoe, A. W. (1986). A timed model for communicating sequential processes. *Proc. Int. Colloquium on Automata, Languages, and Programming, 13th*, pp. 314–321.

Rockstrom, A., and Saracco, R. (1982). SDL-CCITT specification and description language. *IEEE Trans. Comm.* **COM-30** (6), 1310–1317.

Rudin, H. (1982). Validation of a token-ring protocol. *Proc. IFIP Int. Symp. on Local Computer Networks*, pp. 373–387.

Rudin, H. (1983). From formal protocol specification towards automated performance prediction. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 3rd*, pp. 257–272.

Rudin, H. (1984). An improved algorithm for estimating protocol performance. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 4th*, pp. 515–526.

Rudin, H. (1985). An informal overview of formal protocol specification. *IEEE Comm. Magazine* **23** (3), 46–52.

Rudin, H. (1988). Protocol engineering: a critical assessment. *Proc. IFIP Int. Symp. Protocol Specification, Testing, and Verification, 8th*, pp. 3–16.

Rudin, H., and West, C. H. (1982). An improved protocol validation technique. *Computer Networks* **6** (2), 65–73.

Sabnani, K., and Dahbura, A. (1983). A new technique for generating protocol tests. *Proc. ACM/IEEE Data Comm. Symp., 9th*, pp. 36–43.

Sabnani, K., and Schwartz, M. (1984). Verification of a multidestination selective repeat procedure. *Computer Networks* **8**, 463–478.

Saracco, R., and Tilanus, P. A. J. (1987). CCITT SDL: overview of the language and its applications. *Computer Networks and ISDN Systems* **13** (2), 65–74.

Sarikaya, B., and Bochmann, G. (1982). Some experience with test sequence generation for protocols. *Proc. Int. Workshop on Protocol Specification, Testing, and Verification, 2nd*, pp. 555–568.

Sarikaya, B., and Bochmann, G. (1984). Synchronization and specification issues in protocol testing. *IEEE Trans. Comm.* **COM-32** (4), 389–395.

Schindler, S. (1980). Algebraic and model specification techniques. *Proc. Hawaii Int. Conf. on Sys. Sci., 3rd*, pp. 20–34.

Schindler, S., and Steinacker, M. (1979). A formal specification of an X.25 protocol machine. *Proc. NBS Trends and Applications Conf.*, pp. 54–64.

Schindler, S., Didier, J., and Steinacker, M. (1978). Design and formal specification of an X.25 packet level protocol implementation. *Proc. IEEE COMPSAC*, pp. 686–691.

Schwabe, D. (1981). Formal techniques for specification and verification of protocols. Ph.D. dissertation, University of California, Los Angeles, California.

Schwartz, R. L., and Melliar-Smith, P. M. (1981). Temporal logic specification of distributed systems. *Proc. IEEE Int. Conf. on Distributed Computing Systems, 2nd*, pp. 446–454.

Schwartz, R. L., and Melliar-Smith, P. M. (1982). From state machines to temporal logic: specification methods for protocol standards. *IEEE Trans. Comm.* **COM-30** (12), 2486–2496.

Serre, J. M., Cerny, E., and Bochmann, G. V. (1986). A methodology for implementing high-level communication protocols. *Proc. Hawaii Int. Conf. on Sys. Sci, 19th*, pp. 744–754.

Shankar, A. U. (1982). Analysis of communication protocols via protocol projections. Ph.D. dissertation, University of Texas, Austin, Texas.

Shankar, A. U., and Lam, S. S. (1982). On time-dependent communication protocols and their projections. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 2nd*, pp. 215–236.

Shankar, A. U., and Lam, S. S. (1983). An HDLC protocol specification and its verification using image protocols. *ACM Trans. Computer Systems* **1** (4), 331–368.

Shankar, A. U., and Lam, S. S. (1984). Specification and verification of time-dependent communication protocols. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 4th*, pp. 215–226.

Shiratori, N., Takahashi, K., and Noguchi, S. (1988). A software design method and its application to protocol and communication software development. *Computer Networks and ISDN Systems* **15**, 245–267.

Sidhu, D. P. (1982a). Protocol design rules. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 2nd*, pp. 283–300.

Sidhu, D. P. (1982b). Synthesis of communication protocols. *Proc. IEEE Int. Conf. Comm.*, pp. 4C.5.1–4C.5.4.

Sidhu, D. P., and Blumer, T. P. (1984). Automated verification of connection management of NBS class 4 transport protocol. *Proc. ACM SIGCOMM '84 Symp.*, pp. 83–91.

Sidhu, D. P., and Leung, T. K. (1988). Fault coverage of protocol test methods. *Proc. IEEE INFOCOM*, pp. 80–85.

Simon, H. A. (1986). Whether software engineering needs to be artificially intelligent. *IEEE Trans. Software Engineering* **SE-12** (7), 726–732.

Soundararajan, N. (1984). Axiomatic semantics of communicating sequential processes. *ACM TOPLAS*, **6** (4), 647–662.

Sriram, D. (1986). Expert systems for engineering applications. *IEEE Software Magazine* **3** (2), 3–5.

Stalling, W. (1988). "Data and Computer Communications" (second edition). Macmillan, New York.

Stenning, V. N. (1976). A data transfer protocol. *Computer Networks* **1** (2), 99–110.

Sunshine, C. A. (1979). Formal techniques for protocol specification and verification. *IEEE Computer Magazine* **12** (9), 20–27.

Sunshine, C. A. (1982a). Formal modeling of communication protocols. *In* "Computer Networks and Simulation II" (S. Shoemaker, ed.), pp. 53–75. North-Holland, Amsterdam.

Sunshine, C. A. (1982b). Experience with four automated verification systems. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 2nd*, pp. 373–380.

Sunshine, C. A. (1983). Experience with automated protocol verification. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 3rd*, pp. 229–236.

Symons, F. J. W. (1980). The verification of communication protocols using numerical Petri nets. *Australian Telecommunication Research* **14** (1), 34–38.

Tanenbaum, A. S. (1988). "Computer Networks" (second edition). Prentice Hall, New Jersey.

Teng, A. Y. (1980). Protocol construction for computer networks. Ph.D. dissertation, Ohio State University, Columbus, Ohio.

Teng, A. Y., and Liu, M. T. (1978a). A formal approach to the design and implementation of network communication protocols. *Proc. IEEE COMPSAC*, pp. 722–727.

Teng, A. Y., and Liu, M. T. (1978b). A formal model for automatic implementation and logical validation of network communication protocols. *Proc. IEEE Computer Networking Symp.*, pp. 114–123.

Teng, A. Y., and Liu, M. T. (1980). The transmission grammar model for protocol construction. *Proc. NBS Trends and Applications Conf.*, pp. 110–120.

Tenney, R. L. (1983). One formal description technique for ISO OSI. *Proc. IEEE Int. Conf. Comm.*, pp. 1296–1300.

Umbaugh, L. D. (1983). Automated techniques for specification and validation of communication protocols. Ph.D. dissertation, Ohio State University, Columbus, Ohio.

Umbaugh, L. D., and Liu, M. T. (1982). A comparison of communications protocol validation techniques. *Proc. IEEE Int. Conf. Comm.*, pp. 4C.4.1–4C.4.7.

Umbaugh, L. D., Liu, M. T., and Graff, C. J. (1983). Specification and validation of the transmission control protocol using transmission grammar. *Proc. IEEE COMPSAC*, pp. 207–216.

Ural, H., and Probert, R. L. (1984). Automated testing of protocol specifications and their implementations. *Proc. ACM SIGCOMM '84 Symp.*, pp. 148–155.

van Eijk, P. H. J., Vissers, C. A., and Diaz, M., eds. (1989). "The Formal Description Technique LOTOS." North-Holland, Amsterdam, The Netherlands.

Vissers, C. A., and Logrippo, L. (1985). The importance of the service concept in the design of data communications protocols. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 5th*, pp. 3–17.

Vissers, C. A., Tenney, R. L., and Bochmann, G. V. (1983). Formal description techniques. *Proc. IEEE* **71** (12), 1356–1364.

Vuong, S. T. (1983). Reachability analysis of protocols with FIFO channels. *Proc. ACM SIGCOMM '83 Symp.*, pp. 49–58.

Vuong, S. T., and Cowan, D. D. (1982a). A decomposition method for the validation of structured protocols. *Proc. IEEE INFOCOM*, pp. 209–220.

Vuong, S. T., and Cowan, D. D. (1982b). Reachability analysis of protocols with non-FIFO channels. *Proc. IEEE COMPCON Fall '82*, pp. 267–276.

Vuong, S. T., and Cowan, D. D. (1983). UNISPEX—a unified model for protocol specification and verification. *Proc. IEEE INFOCOM*, pp. 318–329.

Vuong, S. T., Hui, D. D., and Cowan, D. D. (1986). VALIRA: a tool for protocol validation via reachability analysis. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 6th*, pp. 35–41.

Walter, B. (1983). Timed Petri nets for modelling and analyzing protocols with real-time characteristics. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 3rd*, pp. 148–160.

West, C. H. (1978a). Generalized technique for communication protocol validation. *IBM. Research and Development* **22** (4), 393–404.

West, C. H. (1978b). An automated technique of communications protocol validation. *IEEE Trans. Comm.* **COM-26** (8), 1271–1275.

West, C. H. (1982). Applications and limitations of automated protocol validation. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 2nd*, pp. 361–372.

West, C. H. (1986). Protocol validation by random state exploration. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 6th*, pp. 233–242.

West, C. H., and Zafiropulo, P. (1978). Automated validation of a communications protocol: the CCITT X.21 recommendations. *IBM. J. Research and Development* **22** (1), 60–71.

Yelowitz, L., Gerhart, L., and Hilborn, G. (1982). Modeling a network protocol in AFFIRM and Ada. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 2nd*, pp. 435–450.

Yemini, Y., and Kurose, J. F. (1982). Can current protocol verification techniques guarantee correctness? *Computer Networks* **6** (6), 377–381.

Yemini, Y., and Nounou, N. (1983). CUPID: a protocol development environment. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 3rd*, pp. 347–356.

Yu, Y. T. (1983). Communicating finite state machines: analysis and synthesis. Ph.D. dissertation, University of Texas, Austin, Texas.

Yu, Y. T., and Gouda, M. G. (1982). Deadlock detection for a class of communicating finite state machines. *IEEE Trans. Comm.* **COM-30** (12), 2514–2518.

Zafiropulo, P. (1978a). Protocol validation by duologue-matrix analysis. *IEEE Trans. Comm.* **COM-26** (8), 1187–1194.

Zafiropulo, P. (1978b). Design rules for producing logically complete two-process interactions and communications protocols. *Proc. IEEE COMPSAC*, pp. 680–685.

Zafiropulo, P., Rudin, H., and Cowan, D. D. (1979). Towards synthesizing asynchronous two-process interactions. *Proc. IEEE Computer Networking Symp.*, pp. 169–175.

Zafiropulo, P., *et al.* (1980). Towards analyzing and synthesizing protocols. *IEEE Trans. Comm.* **COM-28** (4), 651–660.

Zhang, Y. X., *et al.* (1988a). A knowledge-based system for protocol synthesis (KSPS). *IEEE JSAC* **6** (5), 874–883.

Zhang, Y. X., *et al.* (1988b). An interactive protocol synthesis algorithm using a global state transition graph. *IEEE Trans. Software Engineering* **SE-14** (3), 394–404.

Zhao, J. R., and Bochmann, G. V. (1986). Reduced reachability analysis of communication

protocols. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 6th*, pp. 243–253.

Zheng, H. X., and Rayner, D. (1985). The impact of the ferry concept on protocol testing. *Proc. IFIP Int. Workshop on Protocol Specification, Testing, and Verification, 5th* pp. 519–531.

Zic, J. J. (1987). Extensions to Hoare's communicating sequential processes to allow protocol performance specification. *Proc. ACM SIGCOMM '87 Workshop*, pp. 217–227.

Zimmermann, H. (1983). On protocol engineering. *Proc. IFIP Information Processing 83*, pp. 283–292.

Zuberek, W. M. (1985). Performance evaluation using extended timed Petri nets. *Proc. IEEE Int. Workshop on Time Petri Nets*, pp. 272–278.

Zuberek, W. M. (1986). Modified D-timed Petri nets, timeouts, and modeling of communication protocols. *Proc. IEEE Int. Conf. on Distributed Computing Systems, 6th*, pp. 452–457.

This Page Intentionally Left Blank

Computer Chess: Ten Years of Significant Progress

MONROE NEWBORN

School of Computer Science
McGill University
Montreal, Quebec, Canada

Copyright © 1989 by Academic Press, Inc.
All rights of reproduction in any form reserved.
ISBN 0-12-012129-8

1. Introduction

Ten years ago, this writer contributed an article entitled "Recent Progress in Computer Chess" to this series' eighteenth volume (Newborn, 1979). It surveyed developments in computer chess in the middle and late 1970s, developments that raised the playing strength of chess programs to just over the 2000 United States Chess Federation level, the level of a chess Expert. Now, 10 years later, chess programs have improved at least another 500 rating points and are playing almost at Grandmaster level. Grandmasters, of which there are under 50 in North America, hold ratings that begin at approximately 2600 USCF.

The purpose of this article is to describe the technical developments that have led to this remarkably strong level of play. Since 1979, there have been a number of new developments including special-purpose hardware, parallel search on multiprocessing systems, windowing techniques, and increased use of transposition tables. This article describes these advances.

The first working chess programs came into existence in the middle 1950s based on the ideas presented several years earlier by Shannon (1950) and Turing (1953). These programs, developed at the Los Alamos Scientific Laboratory (Kister *et al.*, 1957), IBM in New York (Bernstein *et al.* 1958), and Carnegie-Mellon University (Newell *et al.*, 1958), played very poorly. Some argued, based on the performance of these early programs, that computers would never play strong chess. Gradually, as programmers learned how the search process worked and as computer power increased, programs improved. With programs now on the verge of becoming Grandmasters, and with all signs indicating progress will continue, the day when they will be World Champion cannot be too far off. In Newborn (1979), it was predicted that "with advances in both hardware and software continuing at the same rates as they have during the last 10 years, it is highly probable that programs will be playing Master level chess by 1984, Grandmaster level chess by 1988, and better than any human by 1992. (These are conservative estimates!)" BELLE, in fact, was awarded the title of US Master in 1983, and this year, 1988, as said previously, programs are playing at almost the Grandmaster level. A World Champion by 1992 remains a good bet.

2. Search Techniques in Chess Programs

Chess programs have improved over the years due to the development and refinement of a number of search techniques particularly suited to the capabilities of computers. These techniques are reviewed in the following sections. The minimax algorithm, the foundation of all chess programs, is

examined in Section 2.1. Depth-first search and the basic data structures for chess trees are the subject of Section 2.2. The alpha-beta algorithm, which supplements the minimax algorithm, is presented in Section 2.3 followed in Section 2.4 by material on how moves are generated, how the principal continuation is found and how the killer heuristic is used by chess programs. Section 2.5 describes pruning techniques and variable depth quiescence search. Transposition tables and hashing techniques are presented in Section 2.6. Iterative deepening is introduced in Section 2.7, and the use of search windows is described in Section 2.8. Parallel search techniques are described in Section 2.9. Special-purpose hardware, used by four of the leading programs (BELLE, BEBE, HITECH, and DEEP THOUGHT 0.02) is surveyed in Section 2.10. The problem of using time wisely is considered in Section 2.11.

2.1 The Search Tree and the Minimax Algorithm

For any given chess position, there is a corresponding graph-theoretic game tree in which nodes correspond to positions and branches correspond to moves. The root of the tree corresponds to the position in which a move is to be found. There are typically about 30 to 40 moves in a position. In the initial position, there are exactly 20 moves for White. The rules of chess declare a game drawn if 50 moves pass and no piece has been captured or no pawn has advanced. These two rules imply that the game tree has a finite number of nodes, but the number is astronomical, estimated to be 10^{120}. Thus, except in simple positions, it is impossible to search any more than a small part of the entire game tree.

In the early 1970s, programs searched chess trees at rates of approximately 200 nodes per second. Today, that rate is nearing 1,000,000 nodes per second, an increase by a factor of 5000. The better programs now examine all sequences of moves for approximately eight to ten levels (or plies) in the tree. Certain continuations, those that the program thinks are crucial lines of play, are searched more deeply. To the position at the end of each continuation, called a terminal position, the program assigns a score which is a measure of how good the position is. Functions that assign scores to positions, that is, scoring functions, are much simpler than one might imagine. Shannon (1950) originally proposed a scoring function that took into account material, mobility, and pawn structure. A positive score meant that the computer is winning, a negative one meant that the opponent is ahead. The larger the score, the better the position, and conversely, the more negative the score the worse the position.

Given the scores of these terminal positions, the minimax algorithm provides the rule for determining which move the first player, usually the computer itself, should make at the root of the tree. The minimax algorithm

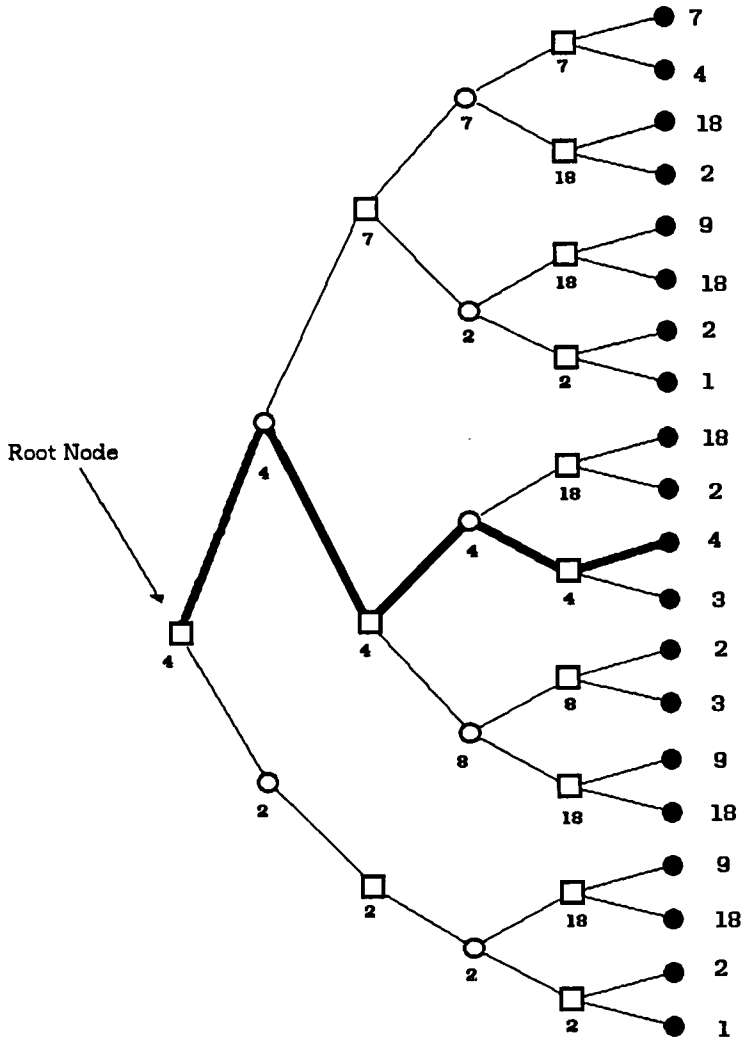


FIG. 1. Five-level game tree showing how minimax algorithm backs up scores to the root. Terminal nodes = ●, non-terminal nodes at which score is maximized = □, non-terminal node at which score is minimized = ○, and principal continuation = □—○—···—●.

says that at even levels in the tree (the root is at level 0), a non-terminal position should be assigned a score equal to the maximum score of any of its successor positions. At odd levels in the tree, the score assigned to a position should be the minimum score of any of its successors. Scores are assigned to nodes by working backward from the terminal nodes toward the root. The

tree in Fig. 1 illustrates this algorithm. Here, a depth-5 search is carried out. There are 20 terminal nodes at level 5, and they are assigned scores by the scoring function as indicated. *Backed-up scores* are assigned to the nonterminal nodes as shown, with a score of 4 being backed up to the root. The computer should thus play the move that leads to the terminal node score of 4. The alternative move leads to a score of 2, not as good for the first player. The sequence of moves leading to the node with the score of 4 is called the *principal continuation*, the continuation that the minimax algorithm calculates is best for both sides to follow.

2.2 Depth-first Minimax Search

All chess programs of any note carry out the minimax algorithm by searching the game tree in a depth-first fashion as opposed to either breadth-first or some sort of best-first search (see Nilsson, 1980). This is mainly because of three reasons. First, depth-first search requires very little memory space. Memory space requirements grow linearly with the depth of search, as opposed to exponentially when breadth-first search or best-first search is used. This was particularly important in the 1960s and the 1970s when memory space was at a premium. Today, with memories measured in megabytes, the advantage of depth-first search over other kinds of search is losing its edge, but as yet, no strong program has been developed that uses anything else. Second, the control strategy used by depth-first search is particularly simple. Deciding where to search next in the tree is well defined; there is no jumping around in the tree as in other types of search. Third, depth-first search can be parallelized more easily than other kinds of search. As a small fourth advantage, printouts of trees produced by depth-first search are easier to interpret in real time than those produced by other types of search.

The flow chart of a depth-first minimax search is shown in Fig. 2. It is based on the data structures shown in Fig. 3. The search calls five subroutines not shown: GENERATE, EVAL, UPDATE, UPDATEPRINC, and RESTORE. The data structures include the following:

1. A representation of the chess board, usually a 64 square array, BOARD(8, 8).
2. An array in which to store the moves at each level in the search tree as they are generated, MOVE (100, 20)—allowing for at most 100 moves in any position and a search to a depth of at most 20 levels.
3. An array of move pointers, MP(20). MP(i) points to the move at level i that is currently under search.
4. An array to keep track of the backed-up scores to nodes in the tree. This array, SCORE(20), has elements SCORE(0),..., SCORE(19).

5. An array to keep track of the principal continuation as it is being formed. This triangular array, PC(20, 20), will have the principal continuation stored in its top row when the search finishes.
6. A stack called CLIST which saves the changes to the board and other data structures when UPDATE is called. RESTORE examines CLIST to see what changes have to be undone when backing up.
7. In addition, the program needs two variables: *DMAX* denotes the maximum depth of search, while *PLY* indicates the current level of the search.

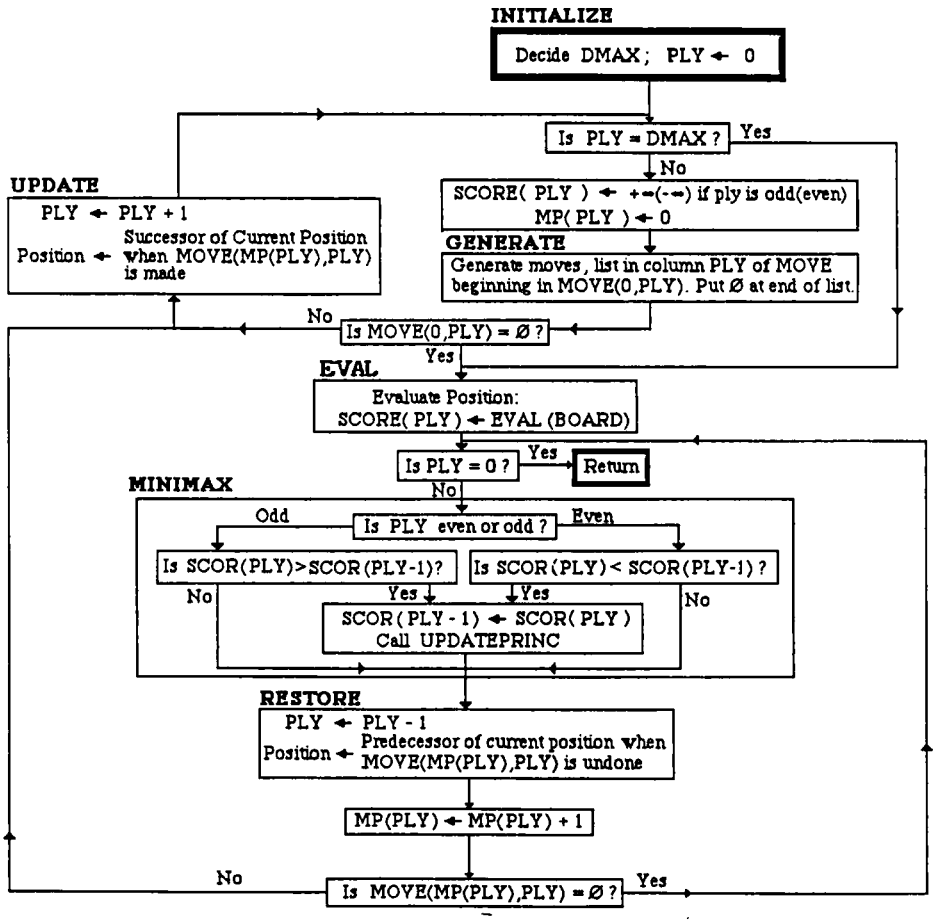


FIG. 2. Flowchart of the depth-first minimax search algorithm.

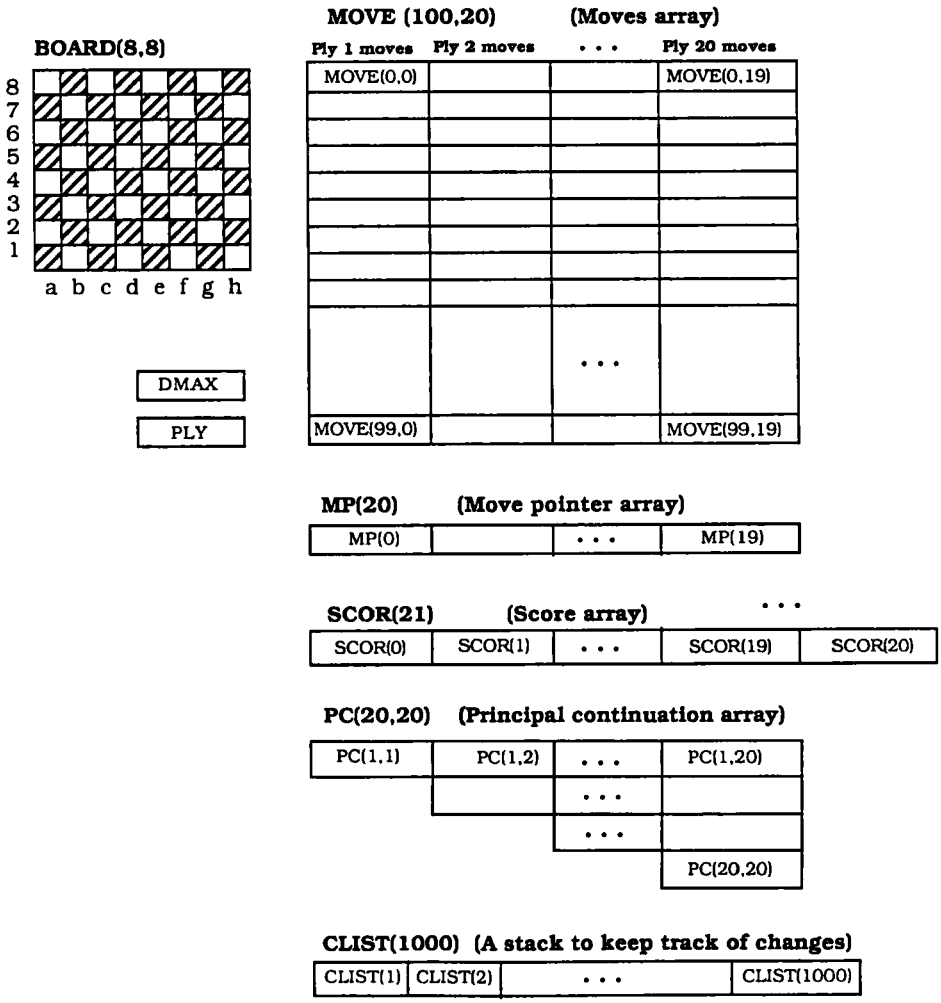


FIG. 3. Basic data structures required by minimax search algorithm.

GENERATE accepts the current board position and the value for *PLY* as inputs. It returns a list of q moves for the position to the MOVES array and places them in MOVES(0, *PLY*), MOVES(1, *PLY*), . . . , MOVES(q-1, *PLY*).

EVAL determines a score for a terminal position. No two programs have the same scoring functions, but most take into account material balance, pawn structure considerations, mobility, king safety, center control (related to mobility), as well as bonuses for trading pieces when ahead, avoiding draws

when playing a weaker player, and mating in as few moves as possible when two or more mating sequences are available. Programs generally assign P = 1, N = 3, B = 3.2, R = 5, Q = 9, and K = 1000, although these values may vary during the course of the game, being conditional on various features of the game. Two Bishops versus two Knights, for example, generally result in a small bonus. Programs quantize features of a position to approximately .01 pawns.

Many factors taken into account by the scoring function can be differentially updated when going from one node to another in the search tree. Material, for example, can be updated when a move is a capture or a promotion. Otherwise material does not change when going from one node in the tree to its successor. Other factors, once computed for one node, can be saved in a hash table and retrieved for use at other nodes. This point is discussed later in Section 2.6.

UPDATE has for its input the current position and the current move under consideration. It updates the board based on the move and saves the changes on the CLIST.

RESTORE has for input the current board and the changes on CLIST that were made to its predecessor to yield the current board. It restores the board to its predecessor.

UPDATEPRINC updates the principal continuation array, PC, when a good move is found. The best move found at level *PLY* along with the sequence of moves that the minimax algorithm calculates the game will follow upon making that move are stored in the PC array in row *PLY* beginning in location PC(*PLY*, *PLY*) as describe in the flow chart in Fig. 4. Essentially, in order to obtain the principal continuation, it is necessary to save the best continuation found thus far in the search at every node on the current

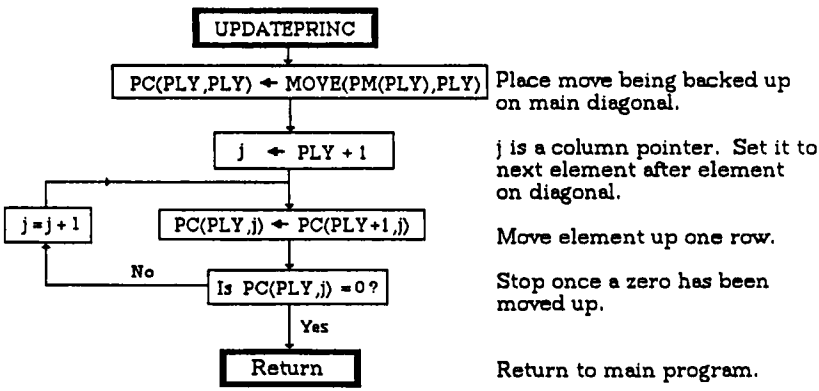


FIG. 4. Algorithm for updating the principal continuation.

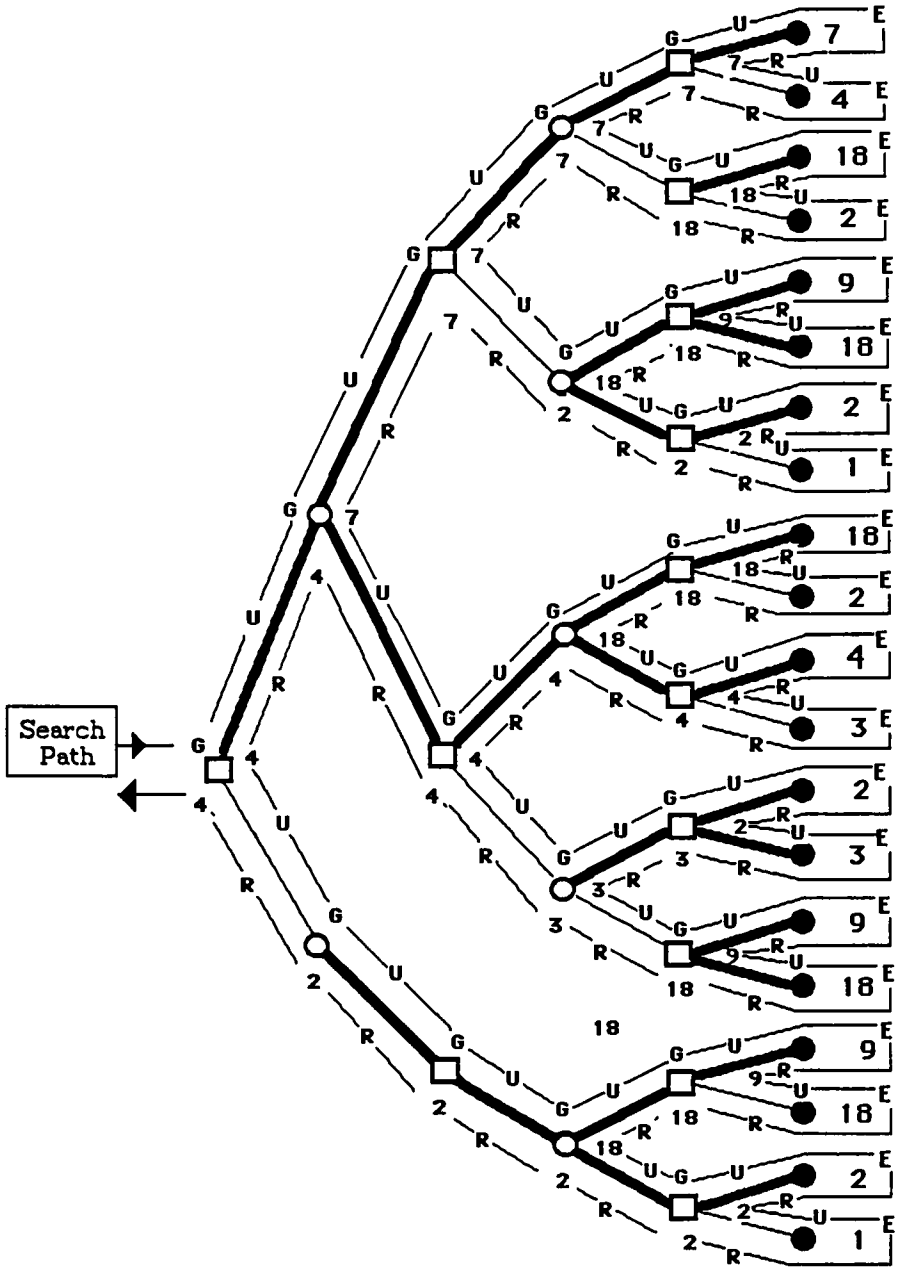


FIG. 5. Minimax algorithm showing where GENERATE, UPDATE, RESTORE, and EVAL take place. Search carried out by computer follows path around tree. GENERATE = G, EVAL = E, UPDATE = U, and RESTORE = R. Bold moves correspond to moves which are the best yet found at a node as the search progresses.

continuation under search. This implies that for a depth D search, D continuations must be kept of lengths D, D-1, D-2,..., 1, and this is done in rows $1, 2, 3, \ldots, D$ respectively of the PC array.

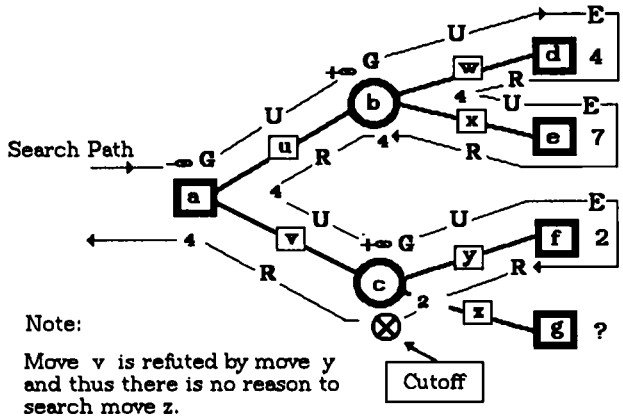
In a depth-first minimax search of the tree shown in Fig. 1 and reproduced in Fig. 5, terminal nodes are scored from top (of the page) to bottom, or as is said, top-down. The figure shows the search path around the tree, beginning and ending at the root. Calls to subroutines are denoted in the figure by the letters G, U, R, and E. Scores are initially assigned to the terminal nodes and backed up to the non-terminal nodes as shown. The search path shows when these events take place. For example, by following the path to the first terminal node which has a score of 7, it can be seen that subroutines G, U, G, U, G, U, G, U, G, U, E were executed. Moves placed in the PC array are indicated by bold lines. The flowchart in Fig. 2 shows that a move is placed in the principal continuation array (UPDATEPRINC is called) whenever a score is backed up from one node to its parent. Note that GENERATE was called 21 times, EVAL was called 20 times, and UPDATE and RESTORE were each called 40 times.

2.3 The Alpha-beta Algorithm

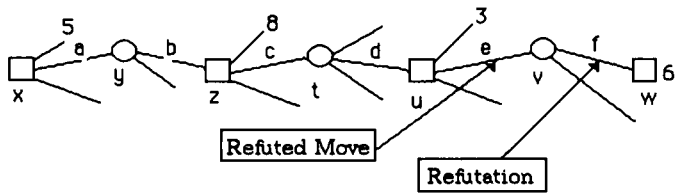
A careful study of the minimax algorithm leads quickly to the realization that there are many paths within the search tree that need not be examined because they have no effect on the outcome of the search. This observation is formalized in the alpha-beta algorithm. Specifically, the alpha-beta algorithm says that once one move at a node *refutes* its predecessor, there is no need to investigate other moves at that node. In the following two paragraphs, a refutation is defined.

Consider the search tree shown in Fig. 6a. Moves are denoted by letters near the end of the alphabet, nodes by letters near the beginning of the alphabet, and scores of nodes by integers. The top-down, depth-first search examines terminal nodes d, then e, and then f. After finding scores for d and e of 4 and 7 respectively, the minimax algorithm assigns a score of 4 to b. The root score can now be bounded from below by 4. Next, node f is scored. Its score of 2 essentially says that no matter what the score of g is, the score of c is at most 2, and given this knowledge, it would be an error to make move v when move u has already been found to lead to a score of 4. And since the score of g is irrelevant, there is no reason to search it. We say that move y is a refutation of move v.

In the general case shown in Fig. 6b, a move x is a refutation of its predecessor move y if the score of node A at even (odd) ply is greater (less) than the score of node C also at any even (odd) ply higher in the tree. A cutoff of search can take place at node B once move x has been searched. The



(a)



Move f refutes move e because the score of node w (6) is less than the backed-up score of node z (8).

(b)

FIG. 6. (a) Example of alpha-beta search, and (b) an example of how deep cutoffs occur in alpha-beta search. In (a), move v is refuted by move y and thus there is no reason to search move z. In (b), move f refutes move e because the score of node w (6) is less than the backed-up score of node z (8).

programming modifications that must be made to the minimax algorithm shown in Fig. 2 to incorporate the alpha-beta algorithm are very minor. For a few lines of code, very large time savings can be achieved. The flowchart of the alpha-beta algorithm presented in Fig. 7 shows that code is added to the flowchart of the minimax algorithm of Fig. 2 in three places. First, two additional elements are necessary for the array SCORE, SCORE(−2), and SCORE(−1), which are initially set to $-\infty$ and $+\infty$, respectively. Second, before generating moves at each node, the score of that node's grandparent is assigned to the node itself. This has the effect of assigning to each nonterminal node at even (odd) ply an initial score equal to the maximum score backed up

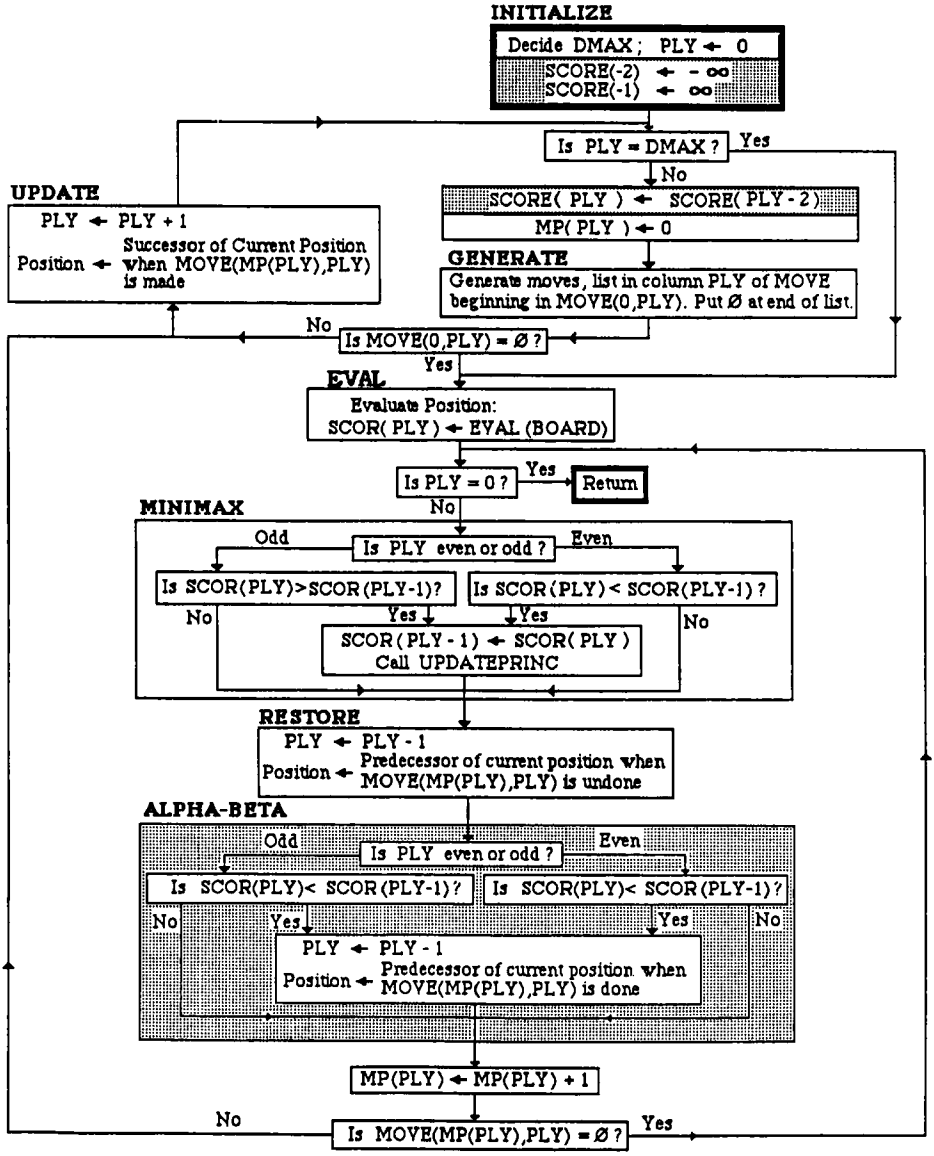


FIG. 7. Flowchart of alpha-beta algorithm. Shaded code has been added to the flowchart of the minimax algorithm in Fig. 2.

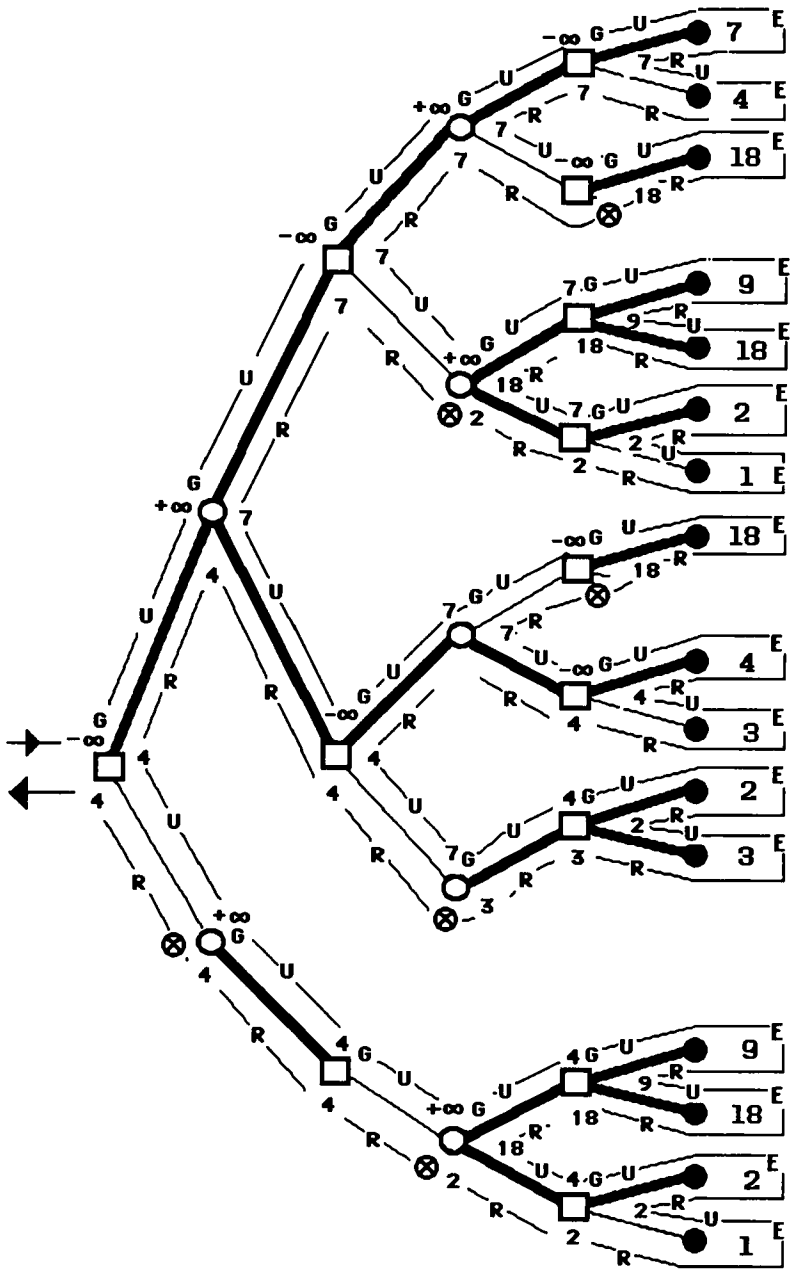


FIG. 8. Alpha-beta search of five-ply game tree showing cutoffs and moves placed in the principal continuation array. GENERATE = G, EVAL = E, UPDATE = U, RESTORE = R, and ⊗ = cutoff. Scores start at $-\infty$ and $+\infty$. Moves in bold are placed in the PC array.

to any node at even (odd) ply higher in the tree. Third, an ALPHA-BETA block is added in which the test for a refutation takes place.

The reader might consider the larger tree shown in Fig. 8. Here an alpha-beta search is carried out on the same tree searched in Fig. 5 by the minimax search. Cutoffs of search are denoted by little circled crosses. Note that there are six cutoffs, although only the first, third, and fourth actually result in reducing the size of the tree searched. Note that EVAL was called 16 times, GENERATE was called 20 times, and UPDATE and RESTORE were each called 35 times.

Slagle and Dixon (1969) showed that for a uniform tree of fanout F and depth D, the number of nodes scored by the alpha-beta algorithm must be at least

$$2F^{D/2} - 1 \qquad \text{(for } D \text{ even)},$$

$$F^{(D+1)/2} + F^{(D-1)/2} - 1 \qquad \text{(for } D \text{ odd)}.$$

Several studies have been carried out on the behavior of the alpha-beta algorithm on models of search trees in which the terminal nodes are assigned random scores. Knuth and Moore (1975) showed that if terminal nodes are assigned random numbers for scores, on average $O(F^2/\log_2 F)$ of the F^2 terminal nodes in a uniform tree of depth 2 are scored. Newborn (1977b) later showed that for games in which terminal node scores were related to the scores of the branches, far fewer terminal node are scored on average. This seemed to better model chess trees where terminal node scores are dominated by material and the material at terminal nodes is dependent on the captures that take place in the search tree. It was shown that in a uniform tree of depth 2, on average $O(F \log_2 F)$ nodes must be scored. For deeper trees the question is still somewhat open for the branch-dependent case, but it could quite well be $O(F^{(D/2)} \log_2 F)$ for trees of arbitrary depth D and fanout F.

In real games, however, terminal nodes are not assigned random scores. Further, a certain amount of information can be gathered during the search that can be used to help order moves at each node from best to worse. The efficiency of the alpha-beta algorithm improves as this ordering improves. In the limit, if moves are ordered from best to worst at each node, the alpha-beta search examines the minimum number of nodes. Several techniques are thus used to locally order moves. They are discussed in the following section.

2.4 Move Generation, the Principal Continuation, and the Killer Heuristic

GENERATE is a sophisticated algorithm in most chess programs. Programs spend more time in this routine than in any other. It is primarily the generation of moves that special-purpose chess hardware is designed to

perform at high speeds. The exact order in which moves are generated and listed is crucial to the speed of the alpha-beta algorithm. It is very important that good moves are listed at the top of the list. With good move ordering, the number of refutations is maximized, and correspondingly the number of cutoffs.

Move generation can be viewed as a two-step process: (1) generate a list of moves, and (2) order the list so that good moves are placed at the top. Programs integrate these two steps to some degree. For example, a move-generation algorithm can generate King moves last in the opening and middlegame, while generating them first in the endgame. What is needed are simple, quick, and reliable algorithms for identifying good moves.

The simplest statistically good moves are captures, and most programs can determine very quickly whether moves are captures and place them at the top of the move list. Bettadapur (1986) showed that captures should be ordered from the capture of the biggest piece to the capture of the smallest piece for best results. Further, the capture of the last-moved piece of the opponent is often a particularly good capture and deserves special placement. Other good moves can be found in the principal continuation array (Akl and Newborn, 1977) and in the killer array.

The killer heuristic (Gillogly, 1972) is used by most chess programs to increase the efficiency of the alpha-beta algorithm. Essentially, moves found to be refutations are kept on a special list called the killer list. Each side has a list of its own. At each node in the search tree when moves are generated, this list is scanned and if one of these moves can be made, it is ordered to the top or near the top of the list of legal moves. Various strategies exist for saving killer moves and using them. By using killer moves, the number of cutoffs is higher than otherwise, and the overall efficiency of the alpha-beta algorithm improves.

Schaeffer (1983) uses the history heuristic to help improve move ordering in his program SUN PHOENIX. This heuristic is a generalization of the killer heuristic. An array of 64 by 64 keeps track of all moves and their effectiveness. This array is used to provide information on the quality of moves as they are generated, using the same philosophy as the killer heuristic. That is, if a move is good in one position, there is a good chance that it is good in another.

2.5 Pruning Techniques and Variable Depth Quiescence Search

When chess programs were first developed they used forward pruning to reduce the effective branching factor at each node (Newell, Shaw, and Simon, 1958). Programs used heuristics to eliminate a high percentage of the moves at each node, hoping that this would allow deeper search along more crucial

and relevant lines. However, forward pruning heuristics were not sufficiently dependable and the programs made horrible blunders. Gradually throughout the 1970s, forward pruning was eliminated from the most successful chess programs.

While forward pruning has been unsuccessful, all the best programs carry out variable depth quiescence searches beyond the arbitrary value *DMAX* set at the beginning of the search. In particular, moves that put the king in check usually require deeper search as do certain capturing moves. Slate and Atkin (1977) report that CHESS 4.9's search tree contains about 50% of its nodes at search depths greater than *DMAX*, and they contend that this is a healthy balance.

For the last year or so, DEEP THOUGHT 0.02 has been using the singular extension heuristic. This heuristic re-searches a move at a node after all other moves at that node are searched if every other move is found to lead to a loss of material. The re-search is done to a greater depth. Singular extensions cause highly forced lines to be searched more deeply than others, and according to Anantharaman *et al.* (1988), this accounts for a significant part of DEEP THOUGHT 0.02's success.

2.6 Transposition Tables

When searching a chess tree containing millions of nodes, many positions are arrived at more than once as a result of a transposition of moves. For example, Fig. 9 depicts a partially-drawn tree rooted at the initial game position. Note that the sequence of moves E2E4, C7C5, D2D4 leads to the same position, say Q, as does the sequence of moves D2D4, C7C5, E2E4. In this case, identical positions result when the moves at the first and third levels of the tree are *transposed*. The five-move sequence E2E3, C7C6, E3E4, C6C5, D2D4 also leads to Q. Of course, so does the four-move sequence E2E4, C7C6, D2D4, C6C5, but in this latter case it is White's turn to move and thus the positions cannot be considered the same. Positions can be considered identical only when, in addition to having pieces on identical squares, castling possibilities, *en passant* possibilities, and whose turn it is are identical.

As the game progresses, a higher and higher percentage of moves transpose, especially King moves. In deep endgames, programs with transposition tables can often find principal continuations 20 plies long and sometimes longer. In Fig. 10, a very deep search is necessary to determine that White should play Kb1. Programs without transposition tables are unable to see how to proceed, while programs with transposition tables are able to do so in less than a minute.

Suppose now that position Q in Fig. 9 had been reached via the first sequence E2E4, C7C5, D2D4 and had been assigned a score as a result of

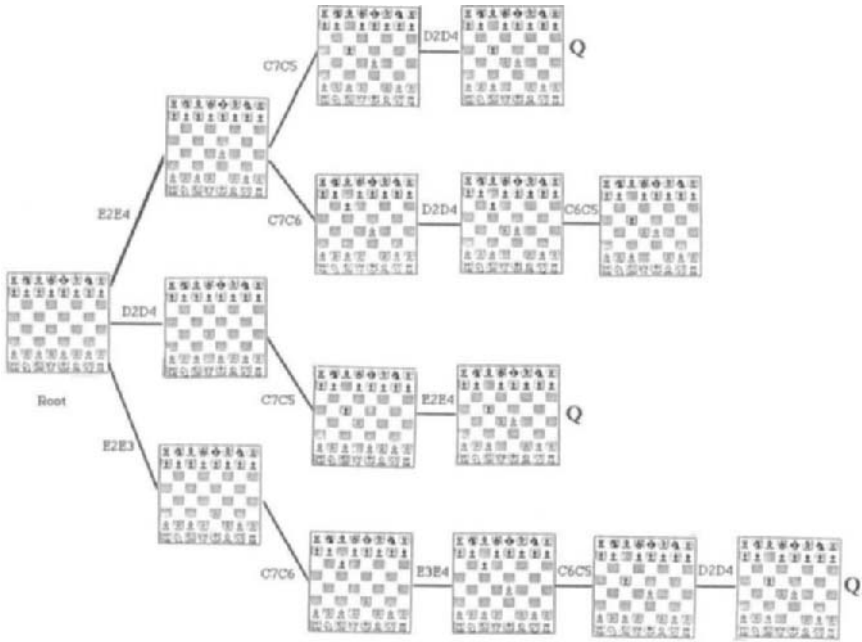


FIG. 9. Partially drawn tree starting at initial position of the game.

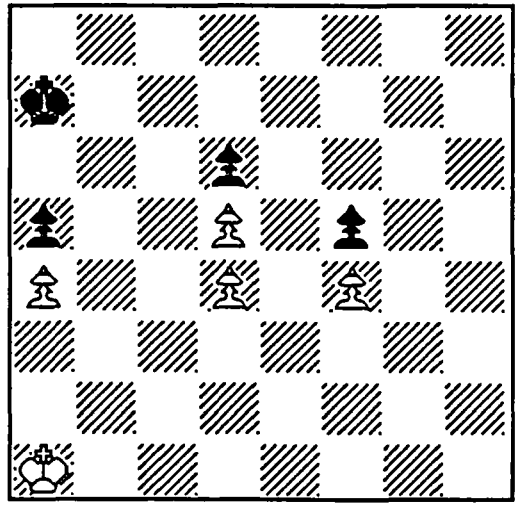


FIG. 10. Kings and pawns endgame position #70 from Fine. White to move.

either scoring it (it would have been scored if the iteratively deepening search was on the third iteration) or searching beyond it and backing up a score to it. Then, when the second sequence D2D4, C7C5, E2E4 arrives at position Q, it is not necessary to search beyond Q or even score Q if the results of the first examination of Q were saved and provided sufficient information. This is what most of the leading chess programs do. Large *transposition tables* are used to save positions and information about them. When search arrives at each position, the transposition table is examined to see whether that position has been reached previously. If it has, and if the information saved about the position is sufficient (what constitutes sufficient information will be discussed shortly), then that position is considered to be a terminal position and assigned a score from the transposition table. Whenever search backs up to a position, the transposition table is search for a match and then the stored information is updated appropriately.

More precisely, when search backs up from some position P, the alpha-beta algorithm has available

1. The score of P which may be an exact value or which may be only an upper or lower bound.
2. The length of the principal continuation rooted at P, denoted by LPC(P).
3. The best move to make in P, denoted BM(P).
4. P, itself, whose turn it is to move, whether *en passant* is possible, and castling possibilities.

The transposition table is searched for a match with P. When found, if the new information gathered about P is better than that currently stored, the entry in the transposition table is updated appropriately. The improved information may be a more precise score or a larger value for LPC(P).

When search arrives at a position, say Y, the transposition table is searched for a match. If there is a match, then Y can be considered a terminal position and assigned the score which was saved in the table if

1. The value of LPC(Y) saved in the table is greater than or equal to $DMAX - \text{ply}$.
2. The score saved in the table entry is exact, or if not exact, the bound on the score is sufficient to cause the move leading to Y to refute its predecessor.

If Y cannot be consided terminal, the move found best the last time Y was searched is available and can be searched first on this try, thus increasing the efficiency of the alpha-beta algorithm.

Figure 11 illustrates how transposition tables can be used to reduce the size of an alpha-beta search. Assume a four-ply search is being carried out.

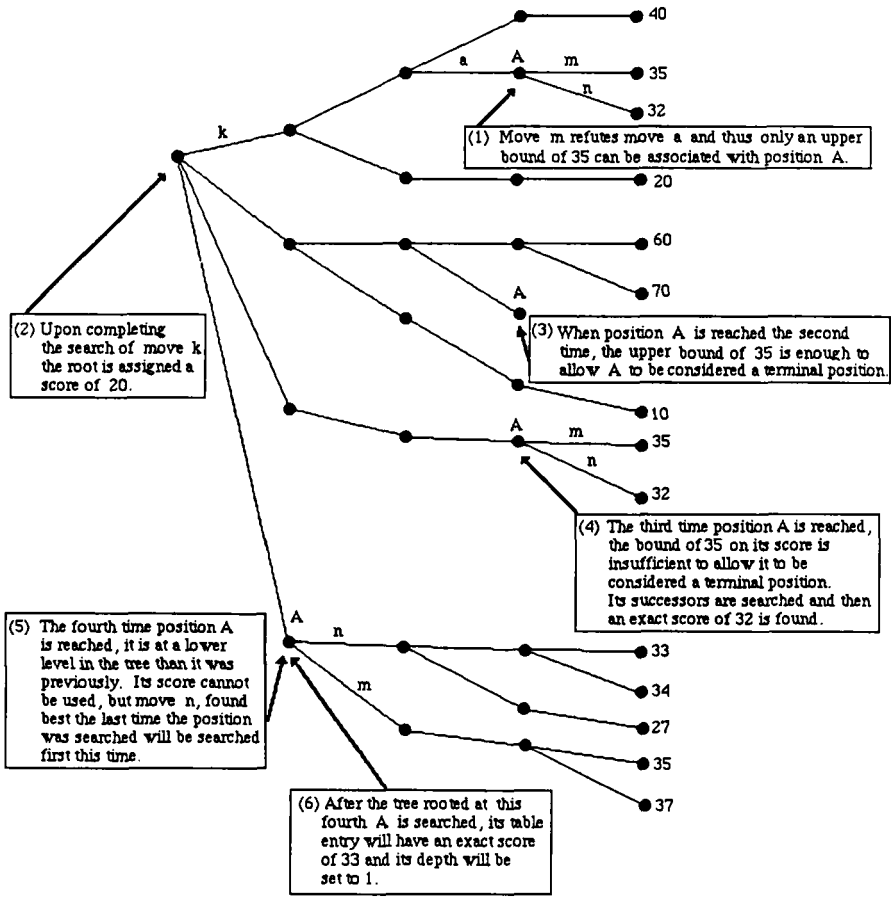


FIG. 11. A tree searched using a transposition table.

Position A arises four times. When search of the first occurence of A has been completed, an upper bound of 35 is assigned to the score of A. When A is reached for the second time, this upper bound of 35 is enough to terminate search. The third time A is reached, the bound of 35 is insufficient to allow A to be considered a terminal position. Its successors must be searched. When they have been searched, a backed-up value of 32 is assigned to A, and move *n* is remembered as the best move to make. The fourth time A is reached, it is only at the first ply, in contrast with the third ply for each of the other three times. The value of LPC saved with A is too small to permit the score obtained previously for A at the third ply to be used to terminate search this time. If it were used to terminate search, that would mean that not all positions in the

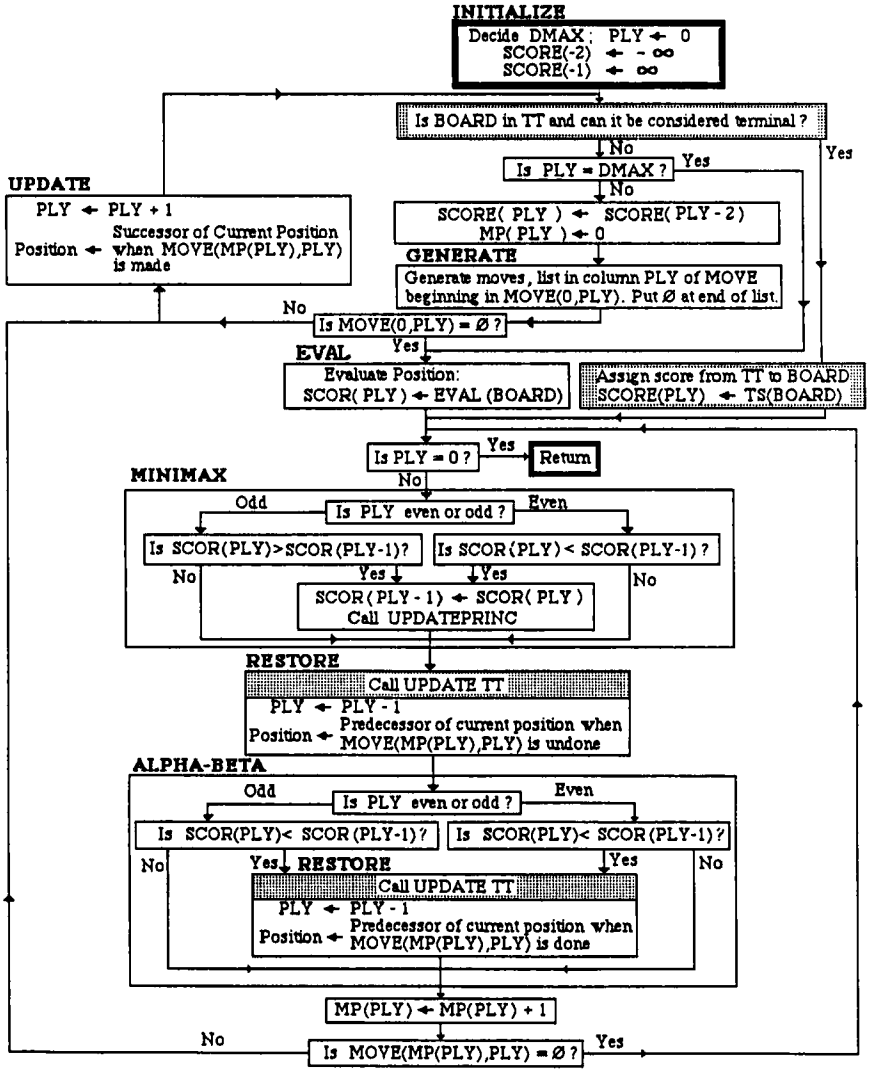


FIG. 12. Flowchart of alpha-beta algorithm using transposition table. Shaded code has been added to the flowchart of Fig. 7.

tree would have been searched to a depth of four plies, contrary to the original objective. Move *n*, however, is stored in the table entry as the best reply in position A, and as the search tree shows, this move is searched first this time. After the tree rooted at the fourth occurence of A has been searched, an exact value of 33 is determined for A.

The flowchart of the alpha-beta algorithm shown in Fig. 7, modified to

handle a transposition table, is shown in Fig. 12. Additional code over and beyond that of Fig. 7 appears in four places. First, when each node is entered, it is checked to see whether it is in the transposition table and, if so, whether it can be considered a terminal node. Second, if this check gives an affirmative answer, then the node is assigned the terminal score found in the table. Third, when search backs up from a node, that node and the relevant information about that node is placed in the transposition table. This happens in two places in the flow table.

In Fig. 13, the effects of using a transposition table on our ongoing five-ply tree are illustrated. Moves are assigned labels and it is arbitrarily assumed that moves that transpose lead to the same positions. That is, move sequence *a-b-c* leads to the same position as does the sequence *c-b-a*. In Fig. 13, three position were found in the transposition table with useful scores. In the case of the position resulting from move sequence *w-c-a*, the transposition table provided a score which permitted search to be terminated without searching the subtree rooted there as was necessary in Fig. 8. This also happened at the node at the end of the move sequence *a-l-f-c*. Note that EVAL was called 10 times, GENERATE was called 16 times, UPDATE and RESTORE were each called 28 times, and the transposition table gave usable scores for three nodes in the tree.

Transposition tables were used in an experiment in machine learning by Slate (1987), perhaps the most significant work done in this interesting area. Only one year earlier, Skienna (1986) reviewed machine learning in computer chess, concluding that "with the exception of rote learning in the opening book, few results have trickled into competitive programs." Slate used a transposition table to store special positions found during the course of a game and to retrieve these positions, when appropriate, in future games. More specifically, positions for which the score changed on the deeper iterations were saved. These positions were considered troublesome for the program. Relative to the total number of positions, these troublesome positions represented a very small percentage. The program might, for example, save in the transposition table a troublesome position which was searched eight plies deep on the twelfth move of the game. Later, in another game on, say, the eighth move, this position might be found at the fourth level in the search tree, and the information learned in previous games and saved in the transposition table would effectively give that move a 12-ply continuation.

Transposition tables usually have 2^k entries, where k ranges from 12 to 24. Each entry can be several words, depending on the word size of the computer and the information that the programmer wishes to save with each position. The number of entries in a transposition table is far less than the number of chess positions. Positions are assigned locations in the transposition table by a *hash function*. A hash function should have two properties: (1) it should randomly spread positions throughout the transposition table, and (2) it should be easy to compute.

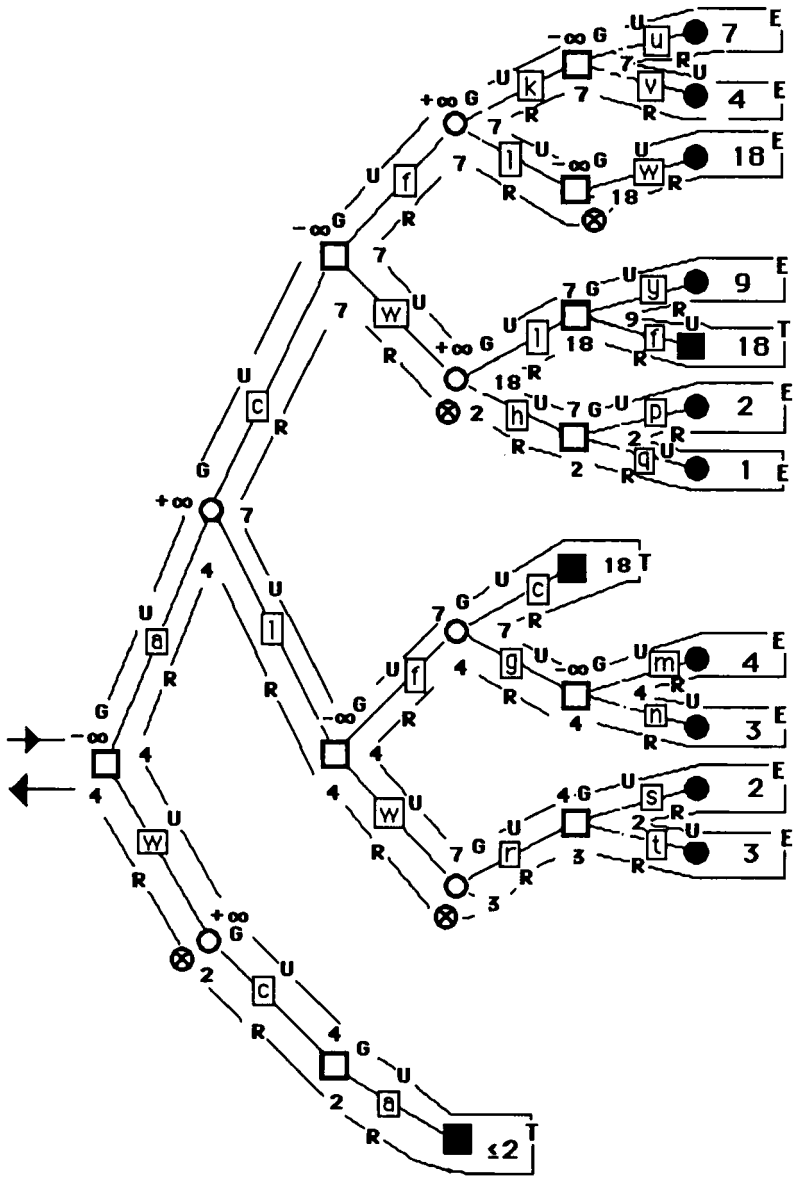


FIG. 13. Ongoing example: five-ply search showing alpha-beta and use of the transposition table. EVAL = E, GENERATE = G, UPDATE = U, RESTORE = R, and TRANS. TABLE HIT = T.

Each position P is assigned a b-bit *hash code* H(P) by a hash function, and then the information about P is stored in or retrieved from the transposition table in the location given by the least significant k bits of H(P). The number of bits b in H(P), of course, must be at least equal to k. During the course of searching a tree, two different positions may have identical values for these k bits assigned by the hash function, a situation that results in a *clash*, whereby two different positions are assigned to the same memory location. To minimize the effect of this type of error, extra information describing a position can be kept with each table entry. If this extra information were a complete description of the position, a clash could always be resolved correctly. However, this is a lot of information. More often, additional bits from the hash code are stored with the table entry as a *key*. A *hashing error* can still occur: two different positions can be assigned the same table entry *and* the same key. Let m denote the number of bits in the key in the following discussion. It should be obvious that the greater the number of bits in the key, the smaller the probability of a hashing error.

Programs have various strategies for resolving what to do when an attempt is made to store a position at a location already occupied by another position with a different key. The program can attempt to store the position at the next location. If that location is found to be occupied also, the program can give up or it can eliminate the older entry of the two that it just examined. Most programs try more than once to find an unoccupied location; CRAY BLITZ tries eight times. Some programs have frequency-of-hits counts saved with each table entry, and entries with low frequency counts are thrown out of the table first when a choice must be made.

Let us assume that a computer with a 32-bit word size is being used by a chess program, that there are 2^{22} words of memory (4 megawords) available for a transposition table, and that each table entry requires two 32-bit words. The table can thus hold information about 2^{21} positions. In the first word of each table entry, 12 bits are used to denote the best move to make in the position, 12 bits to denote the score assigned to the position, two bits to denote whether the score is an upper bound, a lower bound or an exact value, and six bits to denote the level of the position. The 32 bits of the second word are used to store the hash key assigned to the position. This is shown in Fig. 14.

CRAY BLITZ, the current World Champion among computer programs, is described in Nelson (1985), Hyatt *et al.* (1985), and Hyatt (1985). The program uses a six million word transposition table with 64 bits per word. Each table entry requires two words of memory, with the key consuming 40 bits. BEBE uses a home-brewed transposition table with 96-bit words. It can hash up to 256K positions with each position requiring one word: 32 bits for a key, 16 bits for the LPC, 16 bits for the move, 16 bits for a lower bound on the score, and 16 bits for an upper bound.

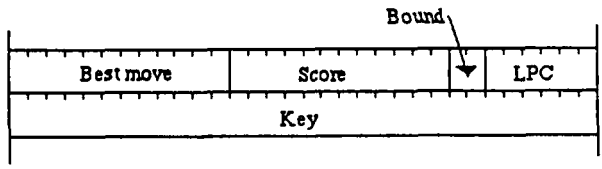
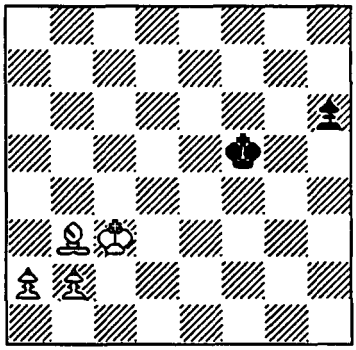


FIG. 14. Data structure for entry in the transposition table.

WK/A1	0110101011101001
WK/A2	0110101011101001
• • •	• • •
WK/C3	1101000101010001
WK/D3	0111110101101011
• • •	• • •
WK/H8	1000010101001001
• • •	• • •
WB/B3	0010001111010101
• • •	• • •
WP/A2	1101011100010101
WP/B2	1000101001001011
• • •	• • •
BK/F5	0100101000100111
BK/G5	1100100101010001
• • •	• • •
BN/C7	0010010111100110
• • •	• • •
BP/H6	1110100101000010

(a) Piece/Square Table



Hash code for this position =

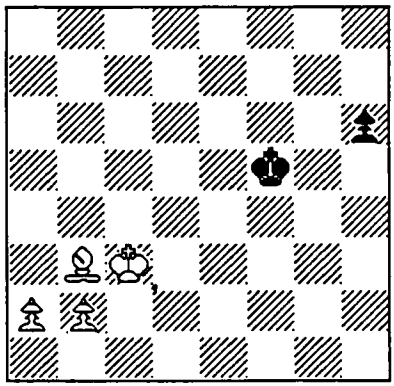
1101000101010001
⊕ 0010001111010101
⊕ 1101011100010101
⊕ 1000101001001011
⊕ 0100101000100111
⊕ 1110100101000010
= 0000110010111111

(b) Encoded chess position

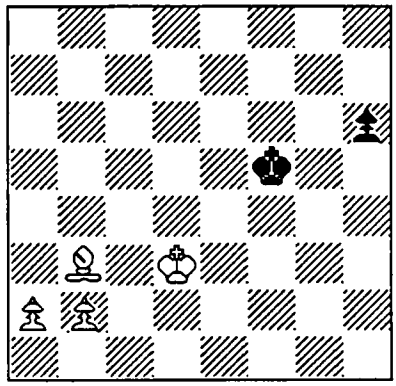
FIG. 15. The assignment of a hash code to a chess position.

Most programs use a hash function similar to the one described by Zobrist (1970). The hash function makes use of a *piece-square table* of 12 (pieces) by 64 (squares) random numbers each of $k + m$ bits. A position is then assigned a hash code by exclusive-or'ing the random numbers assigned to appropriate piece/squares. For example, suppose random numbers are assigned to the piece/squares as shown in Fig. 15a. Then the position shown in Fig. 15b is assigned a hash code as shown. To take into account that positions are different if castling opportunities are different, if *en passant* opportunities are different, or even if whose turn it is is different, extra entries can be included in the piece-square table.

The beauty of this hash function is the ease in computing the hash code of a position that results by making a move in a given position. One removes a piece from one square by exclusive-or'ing the random number of the corresponding piece-square with the hash code for the original position. The piece is then placed on its destination square by exclusive-or'ing the modified hash code with the new piece-square. Small variations on this idea are necessary for capturing moves, queening moves, *en passant* capturing moves, and castling moves. Figure 16 shows how to obtain the new hash code for position that results when C3D3 is played in the position in Fig. 15b.



Hash code for successor position
= 0000110010111111



Hash code for new position when move C3D3 is made
= 0000110010111111
⊕ 1101000101010001
⊕ 0111110101101011
= 1010000010000111

FIG. 16. Illustration of how to obtain a hash code for successor position of Q for the move C3D3.

Hash tables are also used in chess programs for hashing scores for pawn structure and King safety. These hash tables can be much smaller than those for transposition tables. Hit rates observed by Nelson (1985) were quite high resulting in significant savings in time when compared with the alternative of recalculating the value of these factors at each position.

2.7 Iterative Deepening

One of the problems with the early chess programs was deciding exactly how deep to search on each move. It turned out to be a very difficult problem. If the search depth was too shallow, the program made moves too quickly, wasting valuable time. If the search depth was too great, the program might take far more time than reasonable. In the early programs, if the depth setting was too great, some moves at the root of the tree might not get searched at all; the program would stop after some arbitrary time even if it had not searched all first-level moves.

To get around this problem, iteratively deepening searches became popular in the middle 1970s (see Slate and Atkin, 1977). It might be noted that more recently the technique is finding applicability in other problems in artificial intelligence, as discussed by Korf (1985) and Stickel and Tyson (1985). Essentially, rather than carry out one depth-first search to some arbitrarily predetermined depth, a sequence of deeper and deeper depth-first searches are carried out, beginning with a depth of one, then two, and continuing until times runs out. Each iteration finds a principal continuation which is searched first on the next iteration. Each iteration also enters many positions in the transposition table which are used on subsequent iterations. The net result is an improvement in the efficiency of the alpha-beta algorithm, more than compensating for the time required to carry out the extra shallower searches. More importantly, iterative deepening allows search to stop at any time with no serious negative consequences. At worst, when stopping in the middle of the nth iteration, the computer has available the best continuation from the $(n-1)$th iteration. When using iterative deepening, the search is balanced, i.e., every move receives almost equal treatment. Stopping in the middle of an iteration will miss the best move on that iteration only if the best move is ordered below the stopping move on that final iteration. This happens when this best move was also not found best or good enough on the penultimate iteration to warrant being ordered above the stopping move on the final iteration.

2.8 Windows

In the late 1970s, *windows* began to be used in chess programs in conjunction with iterative deepening. Pearl (1980) describes one windowing scheme used by his Scout search algorithm. Programs use windows in different ways, but

what follows is typical. At the beginning of each iteration, an expected root score (RS) is guessed, usually the root score found by the previous iteration. Then on the current iteration, a narrow search window is placed about RS and during the course of the search, continuations that return scores not inside the window are cut off. The *width* of the window is typically two pawns, although some programs use narrower windows. In practice, variables SCORE(-2) and SCORE(-1) are used to keep track of these limits. At the beginning of each iteration, SCORE(-2) and SCORE(-1) are initialized to $RS - P$ and $RS + P$, respectively, where P is the value assigned to a pawn. The window is said to be initialized to $\langle RS - P, RS + P \rangle$. If the guess turns out to be correct, that is, if a move is found with a score within the window when the iteration terminates, search goes on to the next iteration with a revised window again one pawn wide centered about the most recently obtained root score. If no move is found with a score within the window, search is said to fail. Search fails *high* if the score returned is above the window, or *low* if the score returned is below the window. If search fails, the iteration must be repeated to find the true root score and principal continuation. On this *second pass*, the window is determined as follows: If search failed high, the window is set to $\langle RS + P, +\infty \rangle$ to ensure that no second failure will occur. If it failed low, the window is set to $\langle -\infty, RS - P \rangle$. In general, the narrower the window, the faster the search progresses, but the greater the chance of failure. When windows are used, each iteration should be viewed as consisting of two passes, the second pass being unnecessary if the first is successful. The flowchart for this process is shown in Fig. 17.

When using a window of $\langle 0, 8 \rangle$ to search our ongoing example, one additional node is cut off as shown in Fig. 18. The search terminates with success, finding a score of 4 for the root, within the limits of the window. Note that EVAL is called 10 times, GENERATE is called 16 times, and UPDATE and RESTORE are each called 27 times.

There are some improvements that can be made to the windowing strategy described above. First, if some move at the root causes search to fail high on the first pass, search can be stopped immediately and a second pass started with a window $\langle RS + P, +\infty \rangle$. More effectively, the window can be reset to $\langle RS + P, RS + P + 1 \rangle$ and the first pass allowed to continue. If a second move at the root causes search to fail high again, a second pass becomes necessary to determine which move is best. If another high failure does not occur, the best move is known at the end of the first pass although only a lower bound on its score is available. When a second pass is necessary due to two high failures, a re-search is required only of those two moves that caused search to fail high and of those moves ordered lower than the second of these two moves. Generally, it is not necessary to determine the precise score, and thus gambling that the search will not fail high twice is more effective than

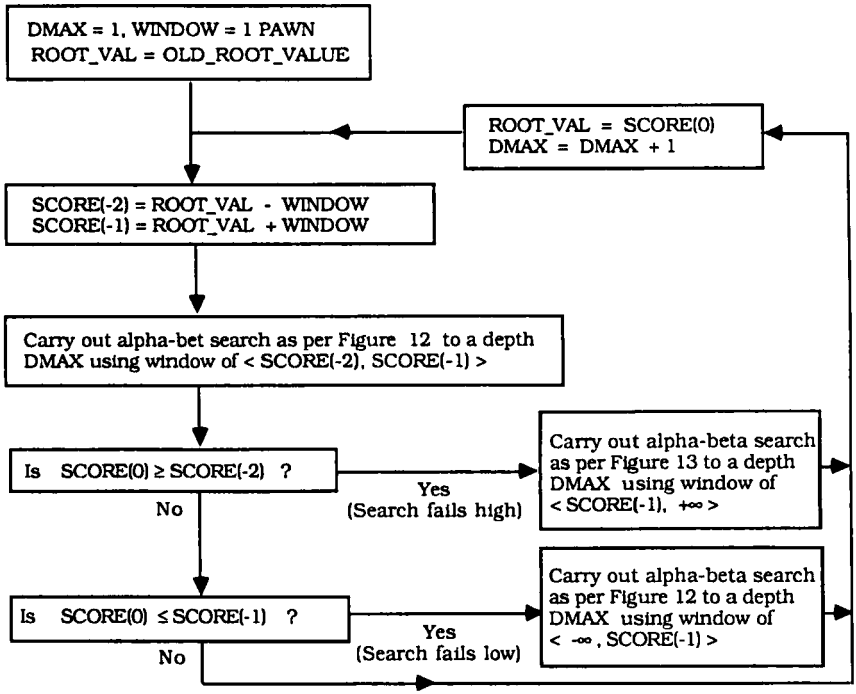


FIG. 17. Flowchart of iteratively deepening search which uses a one-pawn window.

initiating a second pass after one failure. Thompson has used this idea in BELLE for a number of years and is given credit for it by Marsland and Popowich (1985). The process is illustrated in Fig. 19. Suppose moves M1, M2,..., M6 have minimax scores of $+3$, $+18$, $+15$, $+29$, -14, and -2, respectively. The first pass arbitrarily uses a window of $\langle -1, +16 \rangle$. Shaded regions denote subtrees searched. On the first pass, search fails high for the first time when searching M2. The window is raised to $\langle 16, 17 \rangle$ for the remaining moves. M3's subtree, with a score below $+16$, fails low. When searching M4, search fails high for the second time, causing the first pass to terminate. On the second pass, it is not necessary to re-search moves M1 and M3. Futhermore, M4 is searched first and then M2, since M4 failed with a higher score and, knowing only this, is more likely to be the better move. Lastly, M5 and M6 must be searched.

The windowing schemes described above carry out re-searches only at the root of the tree. More sophisticated windowing strategies that allow setting narrow windows and carrying out re-searches of subtrees at all nodes in the tree are used by a number of programs. These recursive procedures are described in papers by Fishburn (1981), Pearl (1980), and Reinfeld *et al.* (1985).

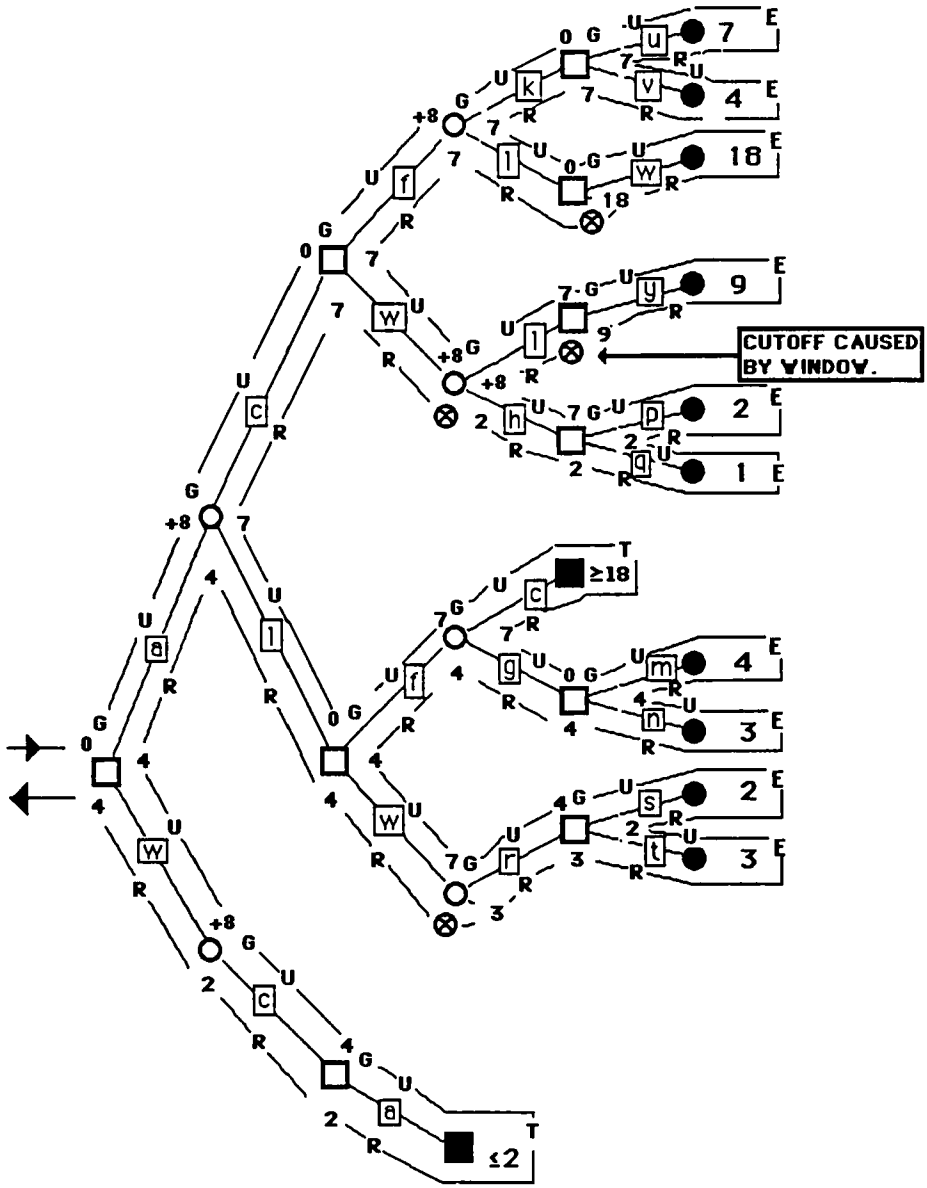


FIG. 18. Five-ply search tree showing alpha-beta cutoffs, effect of transposition table, and window cutoffs. Search window of ⟨0,8⟩ was used.

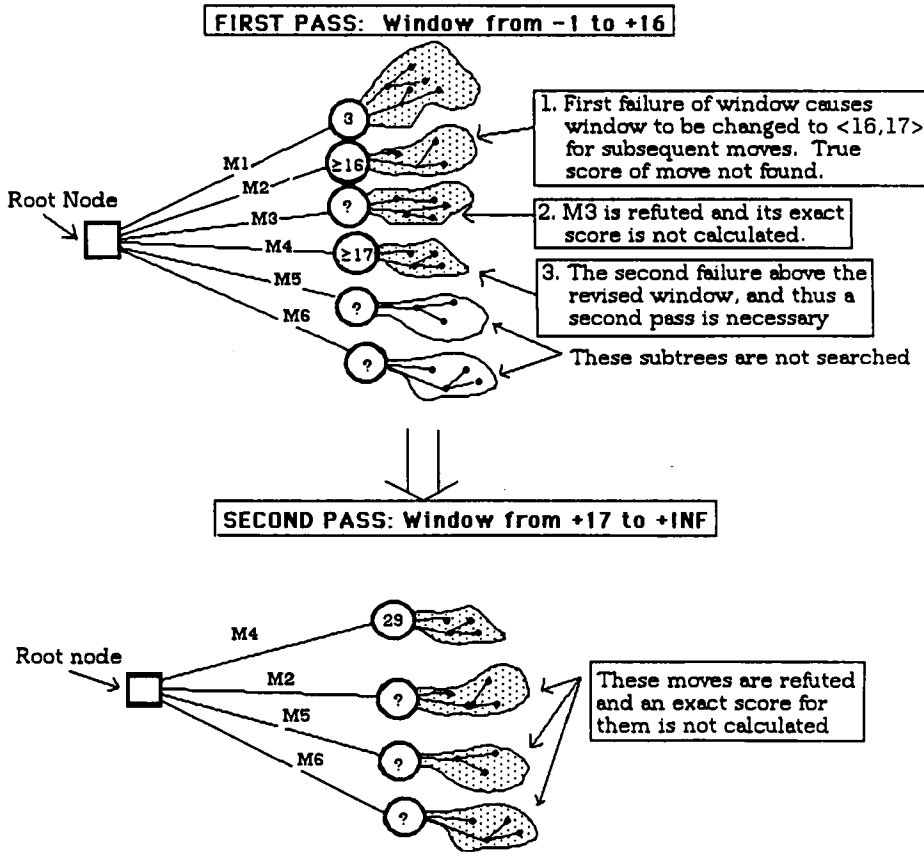


FIG. 19. Two-pass alpha-beta search using windows.

2.9 Parallel Search Techniques

A number of chess programs currently run on multiprocessors. Their objective is to gain an N-fold speedup using N processors. While at first glance this may seem easy to accomplish, in fact, it has been impossible thus far. Nevertheless, impressive results have been recently obtained by WAYCOOL when running on a large NCUBE multiprocessing machine. Felton and Otto (1988) report that they have attained a speedup of 101 on a 256-processor NCUBE and a speedup of 170 on a 512-processor NCUBE.

In 1981, OSTRICH (Newborn, 1982) became the first chess program to compete in a major tournament using a multiprocessing system—five Data General 16-bit computers connected together by a DG communications

package. The system used to principal variation splitting algorithm which is described shortly. In subsequent years, eight DG computers were used.

In 1985, CRAY BLITZ (Hyatt, 1985) was moved onto the four-processor Cray XMP computer, and its programmers also implemented the principal variation splitting algorithm. Later SUN PHOENIX (Schaeffer, 1986) was programmed to run on a network of SUN 3 computers. Again the PVSA was implemented with some variations. Ron Nelson of Fidelity International participated with CHESS CHALLENGER X in the ACM's 17th NACCC in 1986 using 30 microcomputers. WAYCOOL, however, has the distinction of using the most processors to play a game of chess in a major tournament, using 256 processors when participating in the ACM's 19th North American Computer Chess Championship.

The principal variation splitting algorithm is a recursive procedure which is based on iterative deepening (Marsland and Campbell, 1982; Newborn, 1985; Marsland and Popowich, 1985; Marsland *et al.*, 1985). It is illustrated in Fig. 20. On the nth iteration, all processors follow the principal variation found on the $(n-1)$th iteration to the $(n-1)$th level. The tree is dynamically divided up there among all processors. The processors independently search all the moves at that node, and when they finish a final score is determined for the node. Search then backs up one level, where again moves are dynamically divided up and this time two-level subtrees are searched. Eventually, moves at the root are dynamically divided up and the subtrees rooted there are searched. Although interprocessor communication is not particularly a problem, there is considerable waiting time by processors that have no work to do. The scheduling of moves is not sufficiently fine-grained, especially at the root. The granularity problem becomes more pronounced as the ratio of the number of processors to the number of root moves increases. Attempts by Schaeffer (1986) and Felton and Otto (1988) to remedy the inefficiency of the PVSA involve modifying the algorithm to allow more flexible decomposition rules.

Newborn (1988) recently proposed an alternative to the PVSA. The PVSA works well when moves are well-ordered, but in complicated positions where several moves look equally good, the PVSA performs its poorest. A simple alternative called unsynchronized iteratively deepening parallel alpha-beta search was used by OSTRICH beginning in 1985. While on average it does not provide the speedup of the PVSA, it performs quite well in complicated positions. The algorithm works as follows. Carry out two iterations to develop an ordering of root moves, a root score, and a search window for subsequent iterations. Next, distribute the root moves to the processors so that they all get an equal number (maybe differing by one). Then, beginning on the third iteration, have each processor set the narrow window about the expected root score (based on information from the first two iterations) and have them

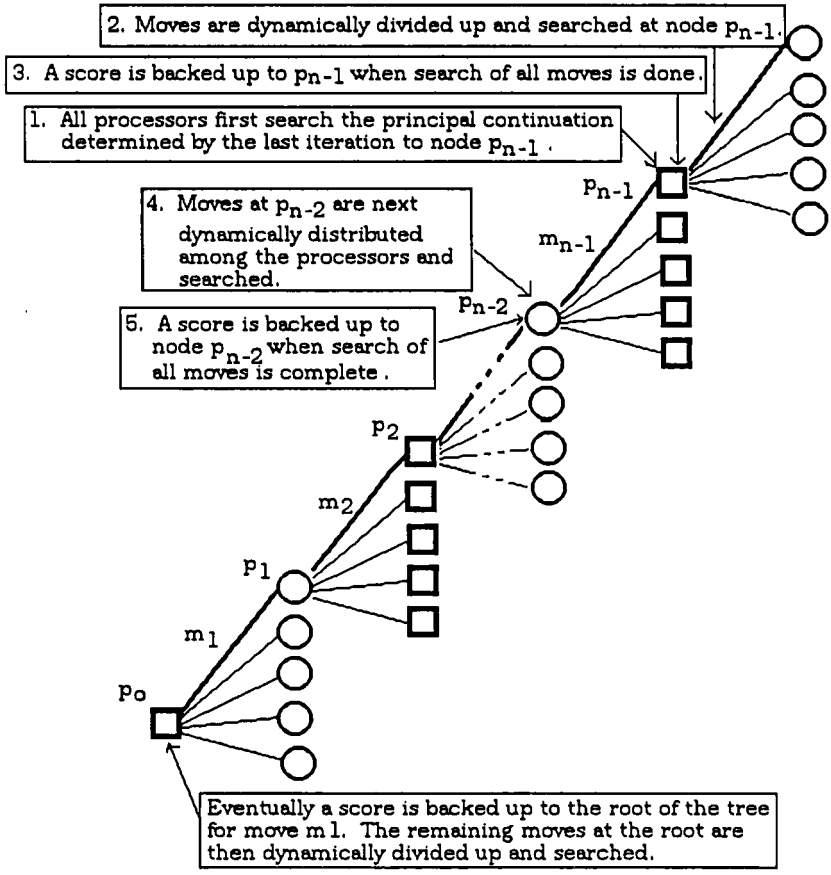


FIG. 20. Partially drawn tree showing how the PVSA divides up search of the tree on the *n*th iteration.

proceed to carry out a sequence of unsynchronized iteratively deepening searches. The use of narrow windows partially compensates for the major relative shortcoming of the algorithm, i.e., not having a provisional root score available as quickly as does the PVSA.

2.10 Special-purpose Hardware

Created in the late 1970s, BELLE (Condon and Thompson, 1982, 1983) serves as the pioneering effort in chess hardware. Three prototypes were built. The first participated in the 1977 World Championship in Toronto. The third won the 1980 World Championship in Linz, Austria. It used several hundred integrated circuits allowing BELLE to search trees at rates in excess of 100,000

nodes per second. BEBE also used special-purpose hardware when participating in the 1980 World Championship. BEBE uses a pipeline approach to generating moves and is able to search about 20,000 nodes/second.

More recently, a group of graduate students at Carnegie-Mellon University under the supervision of Hans Berliner has developed a hardware move generator and special-purpose circuitry to score positions quickly (Berliner and Ebeling, 1986). Carl Ebeling and Andy Palay (1984) did most of the circuit design. Their program, HITECH, searches approximately 200,000 nodes/second. It won the 1987 and 1988 Pennsylvania State Championships, playing against a strong group of human opponents and earning a performance rating of approximately 2400.

A second group at Carnegie-Mellon, incorporating the ideas in BELLE and to a lesser extent in HITECH, has developed the strongest program to date. Initially named CHIPTEST-M, and then renamed DEEP THOUGHT 0.02 in 1988, it runs on a SUN 3 workstation that has a VLSI move generator attached, the first move generator using VLSI technology. The system, developed by Feng-hsiung Hsu (see Hsu, 1986) along with fellow graduate students Thomas Anantharaman, Murray Campbell, Mike Browne, and Andreas Nowatzyk won the ACM's 18th and 19th North American Computer Chess Championships.

FIG. 21. DEEP THOUGHT 0.02's programmer Feng-hsiung Hsu watches Fidelity International's experimental Chess Challenger during their recent game at the ACM's 19th North American Computer Chess Championship.

2.11 Time Control and Thinking on the Opponent's Time

Time-control algorithms are crucial to the success of chess programs. The approach of CRAY BLITZ is described in Hyatt (1984). In tournament play, programs are usually allotted two hours to make the first 40 moves and then an additional hour for the each 20 moves thereafter. This averages three minutes per move. Computers are programmed to take all of this allotted time. For the first few moves, when moves are found in their opening books, moves are made in several seconds. This saved time gets stored up and used later in the game. Most programs have algorithms that force them to take extra time on the first move out of book. They also take extra time when they find their scoring function begins to go negative, or returns a score below expectations. Some are programmed to take less time on certain obvious moves, such as Queen recaptures. Most programs rarely calculate for more than five or six minutes on a move and rarely for less than one minute (unless they guessed their opponent's last move).

Chess programs think about their next move while their opponents are working on their current move. Essentially, they use the principal continuation found on their previous move, assume the opponent will make the second move on that continuation, and then proceed to calculate a reply to

FIG. 22. Robert Hyatt, author of CRAY BLITZ, the current world champion.

the opponent's anticipated move. If they guess incorrectly, they forget what they have done so far and start over. If they guess correctly, they are then in a position to respond immediately to the opponent's move or to continue calculating a while longer. The good programs guess their opponent's move correctly approximately 50% of the time, giving the program the same advantage as running on a computer 50% faster than the one they are running on.

3. Opening Books

Opening books in the better programs contain as many as several hundred thousand positions. BELLE has the largest book, including most of Modern Chess Openings along with countless other lines as well. Most programs that compete in the major tournaments have at least several thousand positions. Opening books help prevent programs from playing openings poorly. Opening theory is very complex, and the scoring functions of most chess programs are not sophisticated enough to avoid greedy play, which can often lead to trouble in the opening. When leaving their opening books, programs often play awkwardly because the book lines leave the programs in positions for which the scoring function is not suited. Great care must be taken to avoid this effect. In particular, many lines used by humans involve a sacrifice of a pawn to gain faster piece development. Programs often are not able to take advantage of the faster development and fail ever to recover from the sacrifice. This may be seen in the game in Section 12.

4. Endgame Play and Endgame Databases

There have been a number of studies specifically on endgame play. There are two general approaches. One is to study endgames in an attempt to understand how expert knowledge can be synthesized and then used. The second approach is at the other extreme: the development of large databases on endgame positions. These databases permit perfect play, although there is no understanding of the principles required to force the win (or draw).

Michie has led much of the effort to study endgame play in the context of expert systems. He is interested in the process of developing rules that allow perfect play if possible, although he settles for strong play if perfect play cannot be achieved. Michie and Bratko (1987) describe rules that can be used to guide play in a KBBKN endgame.

Newborn (1977a) developed a King and pawn endgame program called PEASANT, and studied its effectiveness on a set of positions found in Fine (1941). PEASANT showed that a brute-force search using a simple scoring function could solve a good percentage of the test problems. The effectiveness

of the program would have been much greater if it had had transposition tables.

Thompson (1986) has been the leader in developing databases that allow perfect play in certain endgames. His foremost work has been on the "five-piece endgames," the KBBKN, the KQPKQ, and the KRPKR endgames. In 1977, Thompson came to the world championship in Toronto with a program that played perfect KQKR endgames and took on some of the best players in North America. The players were surprised at how badly they played, being unable to win when they were sure they could. Others have worked on developing endgame databases, most notably KRPKR by Arlazarov and Futer (1978), and KQPKQ by Komissarchik and Futer (1986).

The approach of those building databases is to do so by retrograde analysis as described by Knuth (1973) in the context of the "military game." Starting with positions in which a win exists (either a mate or a move that transforms the position to a won subgame), one works backward, generating predecessor positions. Assuming each side will try to move optimally, each position is assigned a value of win, loss, or draw, and the number of moves to that final outcome.

5. A Brief History of Computer Chess Tournament Play

In 1966, the first recorded match between chess programs took place when a chess program developed at MIT by Alan Kotok (1962) and one developed at the Institute of Theoretical and Experimental Physics in Moscow (see Adelson-Velsky *et al.*, 1970, and Adelson-Velsky *et al.*, 1988) played a four-game match. The Soviet program won two games and drew two others. The games were played by telegraphing moves back and forth across the Atlantic. The match lasted for the better part of a year. In the two games that it won, the Soviet program was searching all moves to a depth of five plies, while in the two games it drew, it was searching all moves to a depth of only three plies. Kotok's program was searching to a depth of four plies in both games, but using unreliable forward pruning.

In 1968, MAC HACK (Greenblatt *et al.*, 1967) became the first chess program that competed in a human tournament. It turned in a respectable performance and earned a rating in the 1400s—the rating of a good high-school player with one year of serious play.

Two years later in New York, the first of the ACM's tournaments was held with six programs participating. Every year since then, the ACM has hosted what was first called the United States Computer Chess Championship, and then renamed the ACM's North American Computer Chess Championship. In 1974, the first World Championship was held in Stockholm, Sweden, as part of the IFIP Congress. That tournament was won by KAISSA, the Soviet

successor of the ITEP program. Since 1974, Soviet programs have been unable to compete successfully with those in Western Europe and North America. This is mainly due to the difficulty of getting computing time and facilities. In the last year, however, the Soviets held a national microcomputer championship, indicating renewed interest on their part.

The results of the World Championships, the ACM Championships, and the World Microcomputer Championships are described in a number of books, including Newborn (1975), Frey (1977), Levy (1976), and Levy and Newborn (1982). The first- and second-place finishers in these events are given in Table I. In addition to the tournaments shown, there have been a number of major tournaments in Europe, in particular, The Netherlands, where computer chess is especially popular.

Programs for microcomputers appeared in the late 1970s. The husband and wife team of Dan and Kathe Spracklen were the leading pioneers. Offsprings of their first program SARGON are the most widely used, commercially available software packages for playing chess. In recent years, they have been developing programs for Fidelity International, Inc., a Miami, Florida-based company. Their programs are used in Fidelity's Chess Challenger series products, the leading chess machine in North America. MEPHISTO, currently the best of the microcomputers by a very narrow margin over Fidelity's best products, was developed by Richard Lang for West Germany's Hegener & Glaser, and it is the most popular chess machine in Europe. David Kittenger's programs used by NOVAG are also quite strong, as are David Levy's programs developed by Intelligent Chess Software in London. The top-line commercial products are playing at the Master level, and soon will be playing at the Grandmaster level.

6. The Rating of Chess Players

The best chess players in the world are given ratings and titles by FIDE, the Fédération Internationale des Echecs. Awarded are the titles of International Master and International Grandmaster. A rating of approximately 2500 and over corresponds to an International Grandmaster, while a rating of approximately 2300 and over corresponds to an International Master. There are currently approximately about 200 International Grandmasters and 1000 International Masters in active competition.

In the United States, the United States Chess Federation gives ratings to its players that correspond closely, but not exactly, to those given by FIDE. USCF ratings are approximately 100 points higher. In the USCF, a Senior Master is rated over 2400 and a Master is rated over 2200. Other nations also rate chess players with the objective of giving ratings that correspond closely to those given by FIDE, as well as giving ratings to players of lesser abilities. In the United States, in addition to Senior Masters and Masters, players are also

TABLE I

RESULTS OF MAJOR COMPUTER CHESS TOURNAMENTS: WORLD CHAMPIONSHIPS, NACCC, AND WORLD MICROCOMPUTER CHAMPIONSHIPS.

World Championships			
Year	City	Winner	Runner-up
1974	Stockholm	KAISSA; Donskoy, Arlazarov, ICL 4/70	CHESS 4.0; Slate, Atkin, CDC 6600
1977	Toronto	CHESS 4.6; Slate, Atkin, CDC Cyber 176	DUCHESS; Truscott, Wright, Jensen, IBM 370/165
1980	Linz	BELLE; Thompson, Condon, PDP 11/23 with chess circuitry	CHAOS; Alexander, Swartz, Berman O'Keefe, Amdahl 470/V8
1983	New York	CRAY BLITZ; Hyatt, Gower, Nelson, Cray XMP 48	BEBE; Scherzer, Chess engine
1986	Cologne	CRAY BLITZ; Hyatt, Gower, Nelson, Cray XMP	HITECH; Berliner, *et al.*, SUN workstation with chess circuitry
ACM's North American Computer Chess Championships			
Year	City	Winner	Runner-up
1970	New York	CHESS 3.0; Slate, Atkin, Gorlen, CDC 6400	DALY CHESS PROGRAM; Daly, King, Varian 620/i
1971	Chicago	CHESS 3.5; Slate, Atkin, Gorlen, CDC 6400	TECH; Gillogly, PDP 10
1972	Boston	CHESS 3.6; Slate, Atkin, Gorlen, CDC 6400	OSTRICH; Arnold, Newborn, DG Supernova
1973	Atlanta	CHESS 4.0; Slate, Atkin, Gorlen, CDC 6400	TECH II; Baisley, PDP 10
1974	San Diego	RIBBIT; Hansen, Crook, Parry, Honeywell 6050	CHESS 4.0; Slate, Atkin, CDC 6400

1975	Minneapolis	CHESS 4.4; Slate, Atkin, CDC Cyber 175	TREEFROG; Hansen, Calnek, Crook, Honeywell 6080
1976	Houston	CHESS 4.5; Slate, Atkin, CDC Cyber 176	CHAOS; Swartz, Ruben, Winograd, Berman, Toikka, Alexander, Amdahl 470
1977	Seattle	CHESS 4.6; Slate, Atkin, CDC Cyber 176	DUCHESS; Truscott, Wright, Jensen, IBM 370/168
1978	Washington	BELLE; Thompson, Condon, PDP 11/70 with chess hardware	CHESS 4.7; Slate, Atkin, CDC Cyber 176
1979	Detroit	CHESS 4.9; Slate, Atkin, CDC Cyber 176	BELLE; Thompson, Condon, PDP 11/70 with chess hardware
1980	Nashville	BELLE; Thompson, Condon, PDP 11/70 with chess hardware	CHAOS; Alexander, O'Keefe, Swartz, Berman, Amdahl 470
1981	Los Angeles	BELLE; Thompson, Condon, PDP 11/23 with chess hardware	NUCHESS; Blanchard, Slate, CDC Cyber 176
1982	Dallas	BELLE; Thompson, Condon, PDP 11/23 with chess hardware	CRAY BLITZ; Hyatt, Gower, Nelson, Cray 1
1983	Not held as the ACM's North American Computer Chess Championship that year but as the Fourth World Championship. See information above on this championship.		
1984	San Fran.	CRAY BLITZ; Hyatt, Gower, Nelson, Cray XMP/4	BEBE; Scherzer, Chess Engine, and FIDELITY EXPERIMENTAL; Spracklen, Spracklen, Fidelity machine
1985	Denver	HITECH; Ebeling, Berliner, Goetsch, Palay, Campbell, Slomer, SUN with chess hardware	BEBE; Scherzer, Chess engine

(*continues*)

TABLE I (*Continued*)

ACM's North American Computer Chess Championships			
Year	City	Winner	Runner-up
1986	Dallas	BELLE; Thompson, Condon, PDP 11/23 with chess hardware	LACHEX, Wendroff, Cray X-MP
1987	Dallas	CHIPTEST-M; Anantharaman, Hsu, Campbell, SUN 3 with VLSI chess hardware	SUN PHOENIX, Schaeffer, Olaffson, 20 SUN 3s
1988	Orlando	DEEP THOUGHT 0.02, Anantharaman, Browne, Campbell, Hsu, Nowatzyk, SUN 3 with VLSI chess hardware	CHESS CHALLENGER X, Spracklen, Spracklen, Nelson, Fidelity machine
World Microcomputer Championships			
Year	City	Winner	Runner-up
1980	London	CHESS CHALLENGER	BORIS EXPERIMENTAL
1981	Travemunde	FIDELITY X	CHESS CHAMPION MARK V
1983	Budapest	ELITE A/S	MEPHISTO X
1984	Glasgow	Four-way tie: ELITE X, MEPHISTO S/X, PRINCHESS, PSION CHESS	
1985	Amsterdam	MEPHISTO AMSTERDAM I	MEPHISTO AMSTERDAM II
1986	Dallas	MEPHISTO DALLAS 3	FIDELITY "2533"
1987	Rome	MEPHISTO	CYRUS 68K
1988	Almeria	MEPHISTO	CHESS CHALLENGER

TABLE II

RATING OF HUMAN CHESS PLAYERS.

Class	USCF Rating Range	Estimated number of active players in the world in this class
Kasparov, Karpov	~2800	2
Senior Master	2400+	400
Master	2200–2499	4000
Expert	2000–2199	40,000
Class A	1800–1999	300,000
Class B	1600–1799	3,000,000
Class C	1400–1599	20,000,000

classed as being Experts, Class A, Class B, and Class C. Other countries have similar categories. Rating ranges are shown in Table II along with the number of players worldwide estimated to be in each class.

7. The Relation Between Computer Speed and Program Strength

For anyone who has ever developed a chess program, computer speed is of paramount concern. Faster computers play better chess. How much faster has been a question for debate since chess programs were first observed in action. This writer (Newborn, 1978, 1979) suggested, based on observations of programs that participated in major tournaments, that over a wide range of ratings, performance seems to improve by about 100 points for every doubling of speed. Since the effective average branching factor of the chess tree is about five or six, this means that each additional level of search improves play by just somewhat over 200 points. Thompson carried out experiments with BELLE shortly thereafter and confirmed these results (Condon and Thompson, 1983; Thompson, 1982). He had seven different versions of BELLE—BELLE(3), BELLE(4),..., BELLE(9)—play 20 game matches against one another and he tabulated the results. The only difference between the seven versions was the depth to which they were set to search: BELLE(i) searched to a depth of i levels. The data obtained by Thompson supported Newborn's 100-point hypothesis for ratings between approximately 1300 and 2000, but the rate of improvement dropped off for higher levels of play. Because of the constraint of time, Thompson carried out his experiment only to search depths of nine plies. Greater search depths, while of particular interest, would have taken large amounts of time to play the 20 game matches.

Subsequently, Newborn (1985) modified his earlier observations in an attempt to reconcile them with Thompson's experimental data. He studied sets of random positions, observing the rate at which programs found improved principal continuations for these positions as search depths ranged from three to 12 plies. He presented a hypothesis that correlated well with Thompson's results over search depths ranging from three to nine and, in addition, allowed one to extrapolate Thompson's results to greater search depths. As search depths increase, Newborn found that a program is gradually less likely to find a better root move than it currently has found, and he observed that this rate correlated closely with rating improvements observed in the range considered by Thompson. He hypothesized that over all search depths, the rate at which the principal continuation is found to change when searching deeper correlates directly with the rating improvement. Thus it is not necessary to play 20 game matches with BELLE(10), BELLE(11),... to determine the ratings of these versions of BELLE. These matches would take great amounts of time with present technology. One can simply test BELLE on a reasonable variety of positions and observe the rate at which the principal continuation changes with increasing search depth. This change will correlate with the rating improvement.

8. On the Chess Skill of Chess Programmers

Early in the development of chess programs, some felt that strong chess players were required to write successful chess programs. However, history has shown this not to be the case. Most of the best chess programs have been written by individuals who are not strong chess players. Furthermore, the programs that they developed turned out to play stronger, sometimes much stronger, than they themselves. Hans Berliner, former World Correspondence Chess Champion is a major exception. Berliner, a strong Master, has developed a program that at this time is also a strong Master. Berliner, however, feels that he is capable of developing a still stronger program. Generally, several individuals have been involved in developing each program, and often one or more of them is a strong player, but not usually the principal one. Shown in Table III is a listing of several prominent programs and approximate ratings of their main programmer as estimated by this writer. The ratings are correct to within approximately 100 points.

Good players, being perfectionist, often hamper the early development of chess programs unless they also have a programmer's mentality. There are a million decisions that have to be made to launch a chess program, and exactly which approach is best is not clear. For example, the board can be represented as an 8×8 array, a 9×9 array or even a 10×10 array. Each representation

TABLE III

RATINGS OF CHESS PROGRAMS AND THEIR PRINCIPAL PROGRAMMERS.

Program	Estimated Rating	Year	Principal author	Estimated rating
DEEP THOUGHT 0.02	2580	1988	Hsu	1200
HITECH	2350	1987	Berliner	2400
CRAY BLITZ	2250	1987	Hyatt	1400
BELLE	2200	1986	Thompson	1700
CHESS 4.9	2100	1980	Slate	2050
KAISSA	1800	1974	Donskoy	1600

has certain advantages and certain disadvantages. In the 8 × 8 representation, it is hard for move generators to determine whether a piece is moving off the edge of the board. In the 9 × 9 representation, the Knight can still jump over the edge. In the 10 × 10 representation, it is easy to determine whether a piece is jumping off the board. However, the added expense of 36 memory locations costs money. It also prevents the board from being stored in the convenient form of an 8 × 8 array, or in some cases, as a linear array of 64 elements. A decision, however, must be made for a board representation, and once it is made, the programmer must live with its consequences for a long time.

A chess programmer must arbitrarily decide on values to assign to pieces. Most assign a value of one to a pawn and so on for the other pieces as discussed earlier. Strong chess players would spend days attempting to refine these values, perhaps making them position-dependent, and the actual writing of the code might never get done.

9. Languages Used by Chess Programs

Most of the best current programs are written in either assembler or C. This includes 21 of the 23 programs that competed in the ACM's North American Computer Chess Championships during the last three years (see Table IV). The other two were written in PASCAL, but neither of these exceptions has participated in the last two years. In fact, for the last two years, the only languages used by programs that have competed in the ACM events have been assembler and C. This widespead use of assembler is something that few would have guessed 20 years ago. At that time it was felt that to develop an expert chess program, a special-purpose chess language was necessary. To date, no such language has appeared. Instead, even FORTRAN seems to be yielding to lower-level languages. The coincidence of having the board being 64 squares,

exactly the size of two computer words, is taken advantage of by all chess programs. Coupled with its great speed, the 64-bit word of the CRAY makes that computer very attractive to chess programmers.

When converting a program from C to assembler, it is possible to obtain a speedup of approximately 20–30%. When converting from other languages to assembler, even more dramatic increases can be obtained. AI Languages such as LISP and PROLOG have not been used by any chess program that participates in major tournaments.

10. Testing Chess Programs

A number of sets of chess positions have become almost standard for testing chess programs. The three most popular are the 300 problems from Reinfeld (1958), the endgame problems from Fine (1941) and the test positions from Bratko and Kopec (1982). The Reinfeld positions are very good for testing the tactical play of programs. Fine's positions test the capabilities of endgame play by programs, while the Bratko/Kopec test set was originally designed to see whether one could distinguish between human style play and that of computers. The Bratko/Kopec set has been used by those interested in testing the efficiency of their search algorithms, in particular those involved in studying the efficiency of various parallel search algorithms.

11. Debugging Chess Programs

Most chess programs, even the best, have bugs. Its only a question of how many and how serious. Thus a large percentage of the time in developing chess programs is spent debugging them. The debugging cycle generally involves observing play until a move is made that does not seem correct, and then rerunning the program on the same position in order to determine why it made the apparently erroneous move. Sometimes, the bug is even more serious. The program might crash in the middle of a long search, and it becomes necessary to find what caused the crash. Most programs have the ability to print the search tree on a terminal or to a file on disk. But the programs are searching several million nodes per move and thus to save the entire tree for latter examination is very awkward and best avoided.

OSTRICH uses a specially designed debugging package for finding errors in the tree. It allows the human debugger to selectively print moves in that part of the tree that is of interest. Moves are printed out when an UPDATE is performed. OSTRICH has two variables, *LEVEL* and *K* which are set before

search begins. *LEVEL* denotes how many levels of the tree to print out. *K* denotes which iteration to begin printing moves. Further, whenever a move is printed out, OSTRICH halts and waits for the human debugger to tell it to proceed. At that point, the user can modify the value of *LEVEL* and *K*: the debugger can increment or decrement *LEVEL* and *K* by 1. Lastly the debugger can print out the current state of the board and other data structures.

For example, suppose the program crashes looking at some node at level four on the fifth iteration. To find out at which node the program crashes, the program can be run on the same position four times. On the first run, set $LEVEL = 1$ and $K = 5$. The program will only print out first-level moves on the fifth iteration, stopping after printing each one, and eventually crashing on one of them, say move M_1. On the second run, again set $LEVEL = 1$ and $K = 5$. When search stops after updating on move M_1 on the fifth iteration, increment the value of *LEVEL*, and then proceed searching. The program will print each reply at level 2 to M_1, stopping after printing each, and eventually crash while looking at one of them, say M_2. The debugger will now know that the program crashed while looking at move M_1 at level 1 and move M_2 at level 2. The third run will yield the three-ply sequence leading to the failure, and finally, the fourth run will lead the debugger to the failing position.

It is often very hard to get a computer to search exactly the same tree when asked to repeat the search of a position. The search of a position depends on many factors in addition to the configuration of pieces on the board. Timing routines can affect how long a position is searched. Draw detection algorithms must be considered. Transposition tables, if they save positions from one move to the next make it virtually impossible to rerun a search and have it be identical to a previous one. When parallel search is used, the debugging problem becomes even more complex.

12. A Sample of Play: DEEP THOUGHT 0.02 (White) Versus HITECH (Black)

The following game was played between DEEP THOUGHT 0.02 and HITECH in the third round of the ACM's 19th North American Computer Chess Championship in Orlando, Florida in November 1988. HITECH had won its first two games and DEEP THOUGHT had a draw and a win and was tied for second place with three other programs. The two programs had played a number of times during the months leading up to this tournament, and this time Berliner got to play an opening line of the Alekhine Defence that he had prepared especially for DEEP THOUGHT 0.02. The opening sacrificed a pawn in return for territory, but HITECH got saddled with weak Kingside

pawns, and pieces that were not sufficiently active. DEEP THOUGHT 0.02 took a clear lead on the 18th move, and except for having to repel a halfhearted counterattack by HITECH, had the game wrapped up after move 21.

DEEP THOUGHT 0.02 was searching between eight and ten plies on most moves, searching trees at a rate of approximately 720,000 nodes per second. A printout of the log of the game created by DEEP THOUGHT 0.02 provided information included below in the analysis of the game. On each non-book move, DEEP THOUGHT 0.02 prints out the first eight moves of the principal continuation and the score of that continuation. DEEP THOUGHT 0.02 anticipated 31 of 56 moves made by HITECH, including all but seven of the moves after the 25th.

1. e4 Nf6 2. e5 Nd5 3. d4 d6 4. Nf3 Nc6 5. c4 Nb6 6. e6 fe 7. Ng5 g6

This move takes DEEP THOUGHT 0.02 out of its opening book.

8. Bd3

Note that 8. Qf3 looks interesting but it just fails: 8. Qf3 e5 (necessary) 9. Qf7 + Kd7 and while Black's King is in an awkward position, Black should be able to recover and maintain its pawn advantage. DEEP THOUGHT 0.02 saw the game continuing 8. ... Nd4 9. Nh7 Nf5 10. Nf7 Rf8 11. Nd2 e5 with a score of −.77 pawns.

8. ... Nd4 9. Nh7 Nf5 10. Nf8 Kf8 11. O-O c5

Black might better have played e5 here, gaining some control of important center squares and giving its pieces, in particular its Queen's Bishop, a bit more freedom.

12. b3 d5 13. Nd2 Qd6 14. Nf3 Nd7 15. Re1 d4 16. Ne5

DEEP THOUGHT 0.02, of course, realizes that this is not a sacrifice. If 16. ... Ne5, then White plays 17. Bf4 pinning the Black Knight to its Queen. White also threatens 17. Ng6. Thus Black is forced to continue:

16. ... Ne5 17. Bf4 Rh7 18. Re5

For the first time, DEEP THOUGHT 0.02's scoring function goes positive. The program expects the game to continue as follows: 18. ... Qb6 19. g4 Nh4 20. Bg3 Bd7 21. Rh5 Rh5.

18. ... Qb6 19. g4 Nh4 20. Bg3

DEEP THOUGHT 0.02 now sees 20. ... Kg8 21. f4 Bd7 22. Qe2 Kg7 23. Rg5 Rg8. But HITECH does not follow DEEP THOUGHT 0.02's line.

20. ... Bd7

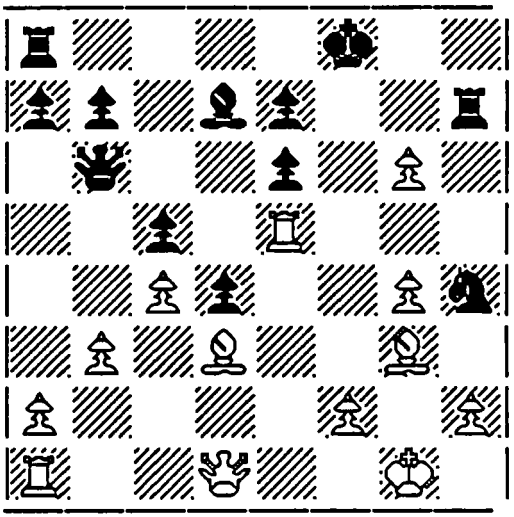


FIG. 23. Position after 20. ... Bd7.

21. Rh5

An elegant move that caught the audience by surprise. DEEP THOUGHT 0.02's scoring function now believes White is ahead by approximately one pawn.

21. ... gh 22. Bh7

DEEP THOUGHT 0.02 now sees: 22. ... Kg7 23. Qd3 e5 24. Bh4 Rh8 25. Bf5 e6, and assigns the continuation a score of +2.69 pawns.

22. ... e5

A good move giving Black's Queen some room to maneuver.

23. Bh4

This time, DEEP THOUGHT sees: 23. ... hg 24. Bg3 Qf6 25. Qd3 b6 26. Re1 Kf7, leading to a score of +2.79 pawns.

23. ... Bg4 24. Qd3 Rc8 25. Re1 Qe6 26. f3 Bh3 27. Qg6

DEEP THOUGHT 0.02 see: 27. ... Qg6 28. Bg6 Rc6 29. Bh5 Re6 30. Bg3 d3, leading to a score of +3.32.

27. ... Qg6 28. Bg6 Rc6 29. Bh5 Re6 30. Bg3 Ra6

HITECH finds a way to hassle DEEP THOUGHT 0.02.

31. a4 d3 32. Re5 Rd6 33. Re1 Rb6 34. Bf4 a5

HITECH has nothing better to do. Black's only chance now is somehow to trade off all material, winning the lone White pawn in the process. This would leave White with a single Bishop, insufficient to mate Black. White, however, is a bit too strong to be led into this scenario. It knows that a lone Bishop is a drawn game.

35. Be3 Rb3 36. Bc5 d2 37. Be7+ Kg7 38. Rd1 Re3
39. Bh4 Ra3 40. Be8 Rf3 41. Bg5 Rf8 42. Bb5 Kg6
43. Be3 Rf3 44. Bd2 Rd3 45. c5 Rd5 46. c6 bc 47. Bc6 Rd6
48. Bf3 Rd4 49. Ba5 Ra4 50. Rd6+ Kf5 51. Bc3 Ra2
52. Rh6 Bg4 53. Bd5 Rc2 54. Rc6 Re2 55. h4 Kf4
56. Rc4+ Kg3 57. Ba5 and Black resigns.

DEEP THOUGHT 0.02 sees the game continuing as follows: 57.... Re7 58. Bc7+ Rc7 (not 58.... Kh4 because of 59. Bd8 pinning the Rook) 59. Rc7 Kh4 60. Rg7.

13. Data on Programs: Computers, Languages, Authors, Affiliations, etc.

Listed below in Table IV are all the programs that participated in the ACM's North American Computer Chess Championships during the last three years.

TABLE IV

PARTICIPANTS IN THE ACM'S 1986, 1987, AND 1988 NORTH AMERICAN COMPUTER CHESS CHAMPIONSHIPS.

Program, Computing System, Language, (Authors and affiliation); Book; Nodes/Sec.

A. I. CHESS! X, IBM-compatible 80286 AT, assembler, 4 mips, (Martin Hirsch, San Francisco); 8K; 2K.

BEBE, SYS-10 Chess Engine, assembler, 65Kb, 16 bits, 10 mips, (Tony Scherzer, Linda Scherzer, SYS-10 Inc., Hoffman Estates, Illinois); 4K; 40K.

BELLE, PDP 11/23 with special chess circuitry, C+microcode, (Ken Thompson, Joe Condon, Bell Laboratories, Murray Hill, New Jersey); 400K; 150K.

BP, Compaq 386, C+assembler, 1Mb, 32 bits, 3–4 mips, (Robert Cullum, Chicago); 8K; 0.5K.

CHESS CHALLENGER X, 28 6502-based microprocessors controlled by a Z-80, assembler, (Ron Nelson, Dan Spracklen, Kathe Spracklen, Danny Kopec, Boris Baczynskyj, Fidelity International, Miami, Florida); 16K+; NA (Participated in 1986).

(*continues*)

TABLE IV (*Continued*)

Program, Computing System, Language, (Authors and affiliation); Book; Nodes/Sec.

CHESS CHALLENGER X, 68030-based microprocessor, (Ron Nelson, Dan Spracklen, Kathe Spracklen, Fidelity International, Miami, Florida): 16K+; NA (Participated in 1988).

DEEP THOUGHT 0.02 (a revised version of CHIPTEST-M, SUN-3 plus high-speed move generator, C, (Thomas Anantharaman, Mike Browne, Feng-hsiung Hsu, Murrary Campbell, and Andreas Nowatzyk, Carnegie-Mellon University, Pittsburgh); 5K; 720K.

CRAY BLITZ, Cray XMP 4/8, FORTRAN + assembler., 128Mb, 64 bits, 105 mips/proc, (Robert Hyatt, Albert Gower, Harry Nelson, University of Alabama, Birmingham); 50K; 100K.

CYRUS 68K, 68020-based microprocessor, assembler, (Mark Taylor, David Levy, Intelligent Chess Software, London, England); 16K; 4K.

FIDELITY X, 68020-based microprocessor, assembler, (Dan Spracklen, Kathe Spracklen, Danny Kopec, Fidelity International Inc., Miami, Florida); 30K; NA.

GNU CHESS, VAX 8650, C, 8 Mb, 32 bits, 6 mips, (Stuart Cracraft, John Stanback, Jay Scott, Jim Aspnes, San Fransisco); 5K; 0.5–1.0K.

GRECO, AT Clone, 16 bits, 1mips, 640Kb, (David Stafford, Dallas, Texas); 1K; 0.45K.

HITECH, SUN 4 with special chess hardware for search and pattern recognition, assembler, (Carl Ebeling, Hans Berliner, Gordon Goetsch, Murray Campbell, Gruss, and Andy Palay, Carnegie-Mellon University); NA, 110K.

LACHEX, Cray XMP 4/16, FORTRAN and assembler, 16mw, 64bits, 105 mips, (Tony Warnock, Burt Wendroff, Los Alamos National Laboratory, New Mexico); 4K; 50K.

MEPHISTO X, 68020-based microprocessor, assembler, 64 Kb RAM, 32 bits, 4 mips, (Richard Lang, Hegener & Glaser A. G., Munich, West Germany); NA; NA.

MERLIN, IBM 3081, PASCAL, 12 mips, (Hermann Kaindl, Marcus Wagner, and Helmut Horacek, Vienna, Austria); 6K; 0.6K.

NOVAG X, 6502 bit sliced microcomputer, 6502 assembler, 4 Kb RAM, 56 Kb ROM, (David Kittinger, Novag Inc., Mobile, Alabama); 22K; 4K.

OSTRICH, 1 DG Eclipse 2/120, 7 DG Nova's 4's, assembler, 64 Kb/proc., 16 bits, 1mips/proc., (Monroe Newborn, McGill University, Montreal); 4K; 2K.

RECOM, 6502 gate array processor, assembler, 8Kb RAM, 8 bits, 4 mips, (Ed Schroder, Deventer, The Netherlands); 7K; 1.5K.

REX III, Intel 80286-based microprocessor, PASCAL, (Don Dailey, Roanoke, Virginia); 0.1K; 0.3K.

SUN PHOENIX, 20 SUN 3 Workstations, C, (Jonathan Schaeffer, Marius Olaffson, University of Alberta, Edmonton); 8K; 20K.

VAXCHESS, Microvas 2, C + assembler, (Tony Guifoyle, Richard Hooker, Hitchen Herts, England); 14K; 1K.

WAYCOOL, 256 proc. NCUBE/10, 1/2 Mb ram/proc., 1 mips/proc., C, (Ed Felton, Steve Otto, Rod Morison, Rob Fatland, Cal Tech, Pasadena, California); NA; NA.

14. The International Computer Chess Association and the ACM's Computer Chess Committee

The International Computer Chess Association was founded in 1977 at the Second World Computer Chess Championship in Toronto. There are currently approximately 700 members from all around the world. David Levy, an International Master from London serves as its president. The first president was Ben Mittman of Northwestern University. Mittman served until 1983 when this writer took over. Levy assumed the position in 1986. The ICCA publishes the foremost journal in the world on the subject of computer chess. Subscriptions are $25(US) and can be obtained by writing to Prof. Jonathan Schaeffer, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada T6G 2H1. The ICCA organizes a world championship every three years. There has been an attempt to alternate these championships between the two sides of the Atlantic. The next championship is scheduled for Edmonton, Alberta in May 1989.

The ACM's Computer Chess Committee was established in the early 1980s with a mandate to coordinate computer chess activities within the ACM. This writer has been the chairman of that committee since its formation. Other members are Ken Thompson, Tony Marsland, Hans Berliner, and Kathe Spracklen.

15. Conclusions

While the last decade has seen programs progress from playing chess at the Expert level to almost that of Grandmasters, the coming decade should be even more exciting for advances in computer chess. It is quite likely that during this period, a computer will defeat the human world champion. There seems to be no limit to the level of play that can be attained by computers, and it seems that the game is sufficiently rich that there will always be room for improvement. The chances are that neither man nor machine will ever discover the optimal way to play the game. Although the level of endgame play by computers is significantly below the level of their middlegame play, it is likely that this will not impede them from becoming better than the best of humans. Their combinational play will give them material advantages in the middlegame that assure victory before the endgame is ever reached.

Where will future improvements occur? Most fundamentally, hardware technology will continue to improve to the advantage of chess programs. There will be an increasing use of multiprocessing systems. Commercial products will soon use multiprocessors. Thousands and eventually—maybe even within the next decade—millions of processors will be used by chess

programs. Special-purpose circuitry will become easier to develop. Opening books will continue to grow in size, and transposition tables will get much larger, as search speeds increase. Improvements in search heuristics will continue to add to the improvement, and increasingly better programming environments will make testing and debugging easier.

How should Grandmasters view these developments? Currently, Grandmasters are studying the games of DEEP THOUGHT 0.02 and HITECH and other leading programs seeking weaknesses in the computers' play. There is nothing unusual about this; all their worthy potential opponents receive this treatment. This puts these programs at a short-term disadvantage since they cannot reciprocate. Grandmasters will find some weaknesses in the programs' inflexible style of play, and they may be able to exploit this shortcoming for the next year or so. But it won't be long before computers become just too good. When that happens, Grandmasters will find they still enjoy the game as they always have, and they will continue competing with one another as well as with their new-found rivals. Those interested in the theory of chess and chess openings, in particular, will use computers as tools.

While Grandmasters will be observing the programs, it will become increasingly important for the programs' authors to become familiar with their opponents' openings and make sure their programs are able to handle them. Learning by chess programs is still a long way off, leaving to the programmers for some time to come the responsibility of updating their programs' books. This will have to be done by carefully following the tournament play of top humans and computers. Eventually, only computer play will be trusted for creating new book lines. Programs may generate their own books during idle time, a development that is inevitable in the coming decade. We may eventually have 14-ply books, 15-ply books, etc., where all moves in the 14-ply book are optimal based on a 14-ply search using the program's scoring function.

For the average chess player who complained in the past about how slowly and poorly programs play, I think you will find this no longer applies. It is now possible for every chess buff to purchase a Master-level program for under two hundred dollars, and that figure is dropping fast. The programs are getting easier to use and are great for teaching young children. My daughter has learned she never has to lose a game. Whenever she observes the program's scoring function go positive, indicating she is losing, she simply changes sides!

References

Adelson-Velsky, G. M., Arlazarov, V. L., Bitman, A. R., Zhivotovsky, A. A., and Uskov, A. V. (1970). Programming a computer to play chess. *Russian Math. Surveys* **25**, 221–262.

Adelson-Velsky, G. M., Arlazarov, V. L. and Donskoy, M. V. (1988). "Algorithms for Games." Springer-Verlag, New York.

Akl, S. G., and Newborn, M. M. (1977). The principal continuation and the killer heuristic. *Proc. Annual Conf. Assoc. Comput. Mach.*, pp. 466–473.

Akl, S. G., Barnard, D. T., and Doran, R. J. (1982). Design, analysis, and implementation of a parallel tree-search algorithm. *IEEE Trans. Pattern Recognition and Machine Intelligence*, pp. 192–203.

Anantharaman, T., Campbell, M., and Hsu, F. (1988). Singular extensions: adding selectivity to brute force searching. *Proc. 1988 American Assoc. Artificial Intelligence Spring Symposium Series*, pp. 8–13.

Arlazarov, V. L., and Futer, A. V. (1978). Computer analysis of a Rook endgame. *In* "Machine Intelligence" **9** (J. E. Hayes, D. Michie, and L. I. Mikulich, eds.). University of Edinburgh Press, Edinburgh, Scotland.

Baudet, G. M. (1978). The design and analysis of algorithms for asynchronous multiprocessors. CMU-CS-78-116, Carnegie-Mellon Univ.

Berliner, H. J. (1986). Computer chess at Carnegie-Mellon University. *In* "Advances in Computer Chess" **4** (D. Beal, ed.). Pergamon Press, pp. 166–180.

Berliner, H., and Ebeling, C. (1986). The SUPREM architecture: a new intelligent paradigm. *Artificial Intelligence* **28**, 3–8.

Bernstein, A., De V. Roberts, M., Arbuckle, T., and Belsky, M. A. (1958). A chess playing program for the IBM 704. *Proc. Western Joint Computer Conf.* **13**, pp. 157–159.

Bettadapur, P. (1986). Influence of ordering on capture search. *International Computer Chess Assoc. J.* **9** (4), 180–188.

Bratko, I., and Kopec, D. (1982). A test for comparison of human and computer performance in chess. *In* "Advances in Computer Chess" **3** (M. R. B. Clarke, ed.). Pergamon Press, pp. 31–56.

Clark, M. R. B., ed. (1977). "Advances in Computer Chess" **1**. University of Edinburgh Press, Edinburgh, Scotland. (Note: Volume 2 was published in 1980. Volume 3 and Volume 4 were published by Pergamon Press, London, in 1982 and 1986, respectively.)

Condon, J. H., and Thompson, K. (1982). Belle chess hardware. *In* "Advances in Computer Chess" **3** (M. R. B. Clarke, ed.). Pergamon Press, pp. 45–54.

Condon, J. H., and Thompson, K. (1983). Belle. *In* "Chess Skill in Man and Machine," second edition (P. Frey, ed.). Springer-Verlag, pp. 201–210.

Ebeling, C., and Palay, A. (1984). The design and implementation of a VLSI move generator. *IEEE 11th Ann. Int. Symp. on Computer Architecture*. Ann Arbor, pp. 74–80.

Felton, E. W., and Otto, Steve W. (1988). A highly parallel chess program. Unpublished manuscript.

Fine, R. (1941). "Basic Chess Endings." David McKay, Philadelphia.

Finkel, R., and Fishburn, J. (1982). Parallelism in alpha-beta search. *Artificial Intelligence* **19**, 89–106.

Fishburn, J. P. (1981). Analysis of speedup in distributed algorithms. Univ. of Wisconsin Tech. Rep. 431.

Frey, P., ed. (1977). "Chess Skill in Man and Machine." Springer-Verlag, New York. (Second edition published in 1983.)

Gillogly, J. J. (1972). The Technology Chess Program. *Artificial Intelligence* **3**, 145–163.

Greenblatt, R. D., Eastlake III, D. E., and Crocker, S. D. (1967). The Greenblatt chess program. *Proc. Fall Joint Computing Conf.*, San Fransisco, pp. 801–810.

Hsu, F. (1986). Two designs of functional units for VLSI based chess machines. CMU Dept. Computer Science Tech. Rep. CMU-CS-86-103.

Hyatt, R. M. (1984). Using time wisely. *International Computer Chess Assoc. J.* **7** (1), 4–9.

Hyatt, R. M. (1985). Parallel search on the Cray X-MP/48. *International Computer Chess Assoc. J.* **8** (2), 90–99.

Hyatt, R. M., Gower, B. E., and Nelson, H. L. (1985). Cray Blitz. *In* "Advances in Computer Chess" **4** (D. Beal, ed.). Pergamon Press, Oxford, pp. 8–18.

Kister, J., Stein, P., Ulam, S., Walden, W., and Wells, M. (1957). Experiments in chess. *J. Assoc. Comput. Mach.* **4,** 174–177.

Knuth, D. (1973). "The Art of Computer Programming" Vol. 1. Addison Wesley, pg. 546.

Knuth, D., and Moore, R. (1975). An analysis of alpha-beta pruning. *Artificial Intelligence* **6,** 293–326.

Komissarchik, E. A., and Futer, A. L. (1986). Computer analysis of a Queen endgame. *International Computer Chess Assoc. J.* **9** (4), 189–200.

Korf, R. E. (1985). Iteratively-Deepening-A*: An optimal admissible tree search. *Proc. Ninth International Joint Conf. Artificial Intelligence*, Los Angeles, California, pp. 1034–1036.

Kotok, A. (1962). A chess playing program for the IBM 7090. B. S. Thesis, MIT. AI Project Memo 41, Computer Center, Cambridge Massachusetts.

Levy, D. N. L. (1976). "Chess and Computers." Batsford Press, London.

Levy, D. N. L, and Newborn, M. M. (1982). "All About Chess and Computers." Computer Science Press, Potomac, Maryland.

Marsland, T. A., and Campbell, M. (1982). Parallel search of strongly ordered game trees. *Computing Surveys* **14** (4), 533–551.

Marsland, T. A., and Popowich, F. (1985). Parallel game-tree search. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 442–452.

Marsland, T. A., Olaffson, M., and Schaeffer, J. (1985). Multiprocessor tree-search experiments. *In* "Advances in Computer Chess" **4** (D. Beal, ed.). Pergamon Press, Oxford, pp. 37–51.

Michie, D., and Bratko, I. (1987). Ideas on knowledge synthesis stemming from the KBBKN endgame. *International Computer Chess. Assoc. J.* **10** (1), 3–13.

Nelson, H. L. (1985). Hash tables in CRAY BLITZ. *International Computer Chess Assoc. J.* **8** (1), 3–13.

Newborn, M. M. (1975). "Computer Chess." Academic Press, New York.

Newborn, M. (1977a). PEASANT: an endgame program for Kings and pawns. *In* "Chess Skill in Man and Machine" (P. Frey, ed.). Springer-Verlag, New York, pp. 119–130.

Newborn, M. (1977b). The efficiency of the alpha-beta algorithm on trees with branch-dependent terminal node scores. *Artificial Intelligence* **8,** 137–153.

Newborn, M. (1978). Computer chess: recent progress and future expectations. *Proc. Jerusalem Conf. Info. Technology*, North-Holland, pp. 216–222.

Newborn, M. (1979). Recent progress in computer chess. *In* "Advances in Computers" **18,** (M. Yovits, ed.). Academic Press, New York, pp. 58–118.

Newborn, M. (1982). Ostrich/P—a parallel search chess program. Tech. Report SOCS-82.3, School of Computer Science, McGill University.

Newborn, M. (1985). A parallel search chess program. *Proc. 1985 ACM Ann. Conf.*, pp. 272–277.

Newborn, M. (1986). An hypothesis concerning the strength of chess programs. *International Computer Chess Assoc. J.* **8** (4), 209–215.

Newborn, M. (1988). Unsynchronized iteratively deepening parallel alpha-beta search. *IEEE Trans. Pattern Analysis and Machine Intelligence* **10** (5), 687–694.

Newell, A., Shaw, J. C., and Simon, H. A., (1958). Chess playing programs and the problem of complexity. *IBM J. Research & Development* **4** (2), 320–335.

Nilsson, N. J., (1980). "Principles of Artificial Intelligence." Tioga Press, Palo Alto, California.

Pearl, J., (1980). Asymptotic properties of minimax trees and searching procedures. *Artificial Intelligence* **14,** 113–138.

Reinefeld, A., Schaeffer, J., and Marsland, T. (1985). Information acquisition in minimal window search. *Proc. Ninth International Joint Conf. Artificial Intelligence, pp.* 1040–1043.

Reinfeld, F. (1958). "Win at Chess." Dover Publications Inc., New York.

Schaeffer, J. (1983). The history heuristic. *International Computer Chess Assoc. J.* **6** (3), 16–19.

Schaeffer, J. (1986). Improved parallel alpha-beta search. *1986 Proc. FJCC*, pp. 519–527.

Shannon, C. E. (1950). Programming a computer for playing chess. *Philosophical Magazine* **41,** 256–275.

Skienna, S. S. (1986). An overview of machine learning in computer chess. *International Computer Chess Assoc. J.* **9** (1), 20–28.

Slagle, J. R., and Dixon, J. K. (1969). Experiments with some programs that search game trees. *J. Assoc. Comput. Mach.* **16,** 189–207.

Slate, D. J. (1987). A chess program that uses the transposition table to learn from experience. *International Computer Chess Assoc. J.* **10** (2), 59–71.

Slate, D. J., and Atkin, L. R. (1977). CHESS 4.5—The Northwestern University Chess Program. *In* "Chess Skill in Man and Machine" (P. Frey, ed.). Springer-Verlag, pp. 82–118.

Stickel, M. E., and Tyson, W. M. (1985). An analysis of consecutively bounded depth-first search with applications in automated deduction. *Proc. Ninth International Joint Conf. Artificial Intelligence*, Los Angeles, California, pp. 1073–1075.

Thompson, K. (1982). Computer chess strength. *In* "Advances in Computer Chess" **3** (M. Clarke, ed.). Pergamon Press, pp. 55–56.

Thompson, K. (1986). Retrograde analysis of certain endgames. *International Computer Chess Assoc. J.* **9** (3), 131–139.

Turing, A. M. (1953). Digital Computers applied to games. *In* "Faster than Thought" (B. V. Bowden, ed.). Pitman, London, pp. 286–310.

Zobrist, A. L. (1970). A hashing method with applications for game playing. Tech. Rep. 88, Computer Sciences Dept., University of Wisconsin, Madison, Wisconsin.

Soviet Computers in the 1980s: A Review of the Hardware

Richard W. Judy and Robert W. Clough*

Hudson Institute
Herman Kahn Center
Indianapolis, Indiana

1. Introduction

Soviet computing in the 1980s has become a very interesting scene. This has been the decade when the nation's top political leadership finally recognized the central role of computers and other information technologies in military, economic, and social development. But that recognition came very late in the day and not before the Soviet Union's international competitors had attained a huge, perhaps insurmountable, lead in both the technologies and their applications.

It would be wrong to suppose that the Soviets have made no progress in computer technology. As this paper demonstrates, there has indeed been progress. But it has been progress of an *absolute* variety, or one relative to the previously underdeveloped state of this technology in the USSR. Compared

* The views expressed herein are the authors' own. The research reported for this paper was supported in part by the National Council for Soviet and East European Research. The authors gratefully thank Virginia Clough of the Hudson Institute and Steven Flinn of Indiana University for their vital assistance.

Copyright © 1989 by Academic Press, Inc.
All rights of production in any form reserved.
ISBN-0-12-012129-8

with Western and Japanese progress in developing and using information technologies of all kinds, the Soviet Union has continued to lose ground rapidly in the 1980s.

The lengthening qualitative lag will be obvious to the discerning reader. But the quantitative lag is also huge. Total Soviet production of all computers except personal computers (PCs) was stable from 1985 to 1987 at about 16,000 units per year. Production of PCs, where the Soviet lag is most pronounced, amounted to merely 8800 units in 1985. In 1986, PC output had risen to 27,600 and in 1987, to 51,200.[1] Total PC production in the 12th Five Year Plan (1986–1990) was originally planned to reach 1.1 million units. That target was recently slashed to 500 thousand.[2]

The purpose of this paper is to provide a reasonably comprehensive survey of the important Soviet civilian computer hardware produced in the 1980s. This is done against a backdrop of previous developments that is sketched in Section 2. Section 3 quickly summarizes the official plans for computer technology in the 1980s.

Section 4 is the heart of the paper. Here we discuss the main computer systems that have been designed or manufactured in this decade. The exposition differs somewhat from that usually encountered in Western discussions of Soviet computers in that we have organized it not by type of computer but, rather, by manufacturing or designing organization. If this were a treatise on American computer hardware, the reader would hardly be surprised if the exposition were organized by manufacturer, e.g., IBM, DEC, or Apple. Contrary to the impression often conveyed in the Western literature, the Soviet computer industry is far from monolithic. A peculiar kind of competition prevails among its principal players. But is it a healthy competition? We believe that an appreciation of the differing roles played by the large manufacturing ministries and the Academy of Sciences is necessary for a proper appreciation of the state of Soviet computer technology today. It is even more necessary for an understanding of *why* matters have come to be as they are.

Finally, Section 5 takes a look at the impact of Mikhail Gorbachev's reforms (*perestroika*) on the Soviet computer hardware industry. The purpose in doing this is to see what promise recent policy changes may hold for bringing needed improvements.

Whatever its problems, the Soviet computer scene has become too expansive to encompass in a single article. In a sequel to be published in this series, we intend to survey and evaluate Soviet computer software and applications developments in the 1980s.

[1] *Vestnik Statistiki* (1988) (7), 62.

[2] Velikhov (1988), 26.

2. Soviet Computing Before 1980: A Brief Summary

The history of Soviet computing begins in the Institute of Electronics at the Ukrainian Academy of Sciences in Kiev. There, in 1947, Academician Sergei Alekseevich Lebedev founded a special laboratory to design an electronic digital computer. In 1950, this design was completed and, in 1951, took form as the MESM (*Malaia Elektronaia Schetnaia Mashina*)[3], the first Soviet electronic computer. It was there in the period 1946–1951 that a core group of future Soviet computer scientists was formed.[4]

In 1950, Lebedev organized a new laboratory for computer design in the Institute of Precise Mechanics and Computer Engineering of the USSR Academy of Sciences ("IPMCE") in Moscow. IPMCE was then headed by Academician Mikhail Alekseevich Lavrent'ev who later became President of the Siberian Division of the USSR Academy of Sciences where he strongly encouraged the development of computer sciences. Lebedev moved his residence to Moscow in 1951. Two years later, he became head of IPMCE where he continued to guide computer research and design, and to train young computer scientists, until his death in 1974. Lebedev's place in Kiev was taken by Viktor Mikhailovich Glushkov who guided what eventually became the Institute of Cybernetics until his untimely death in 1982.

At IPMCE, the Lebedev group set about designing a large-scale computer, the BESM-1 (*Bol'shaia Elektronaia Schetnaia Mashina*)[5] which was approved by a state acceptance commission headed by Lavrent'ev in 1953. Serial production of a somewhat modified version of this machine using ferrite core memory, the BESM-2, began in 1958. As in the United States, the earliest Soviet computers (e.g., MESM and BESM) were involved in scientific and military computing, especially in computing centers organized at various locations of the Academy of Sciences.

Automatic data processing for "commercial" purposes (ADP) made its Soviet debut in the mid 1950s with the appearance of the MINSK and URAL designs. The MINSK was designed by V. V. Przhiialkovskii and others in Minsk at the design bureau of the Ordzhonikidze factory belonging to the Ministry of the Radio Industry (Minradioprom). The URAL was designed by B. I. Rameev in Penza. Both the MINSK and URAL were manufactured by Minradioprom and they became the workhorses of Soviet ADP.

In the early 1960s, versions of BESM, MINSK and URAL machines using solid-state circuitry and ferrite core memory made their appearance. A variety

[3] "Small Electronic Calculating Machine."

[4] For more on the history of Soviet computing, see Burtsev (1985); Campbell (1976); Davis and Goodman (1978); Ershov (1975); Glushkov (1979); Judy (1967); Judy (1970); Korolev and Mel'nikov (1976); Mel'nikov (1986); Rudins (1970); and *USiM* (1976, 1977, 1982).

[5] "Large Electronic Calculating Machine"

of other machines also appeared in the 1960s, but the Lebedev machines dominated the field of scientific computing, while Minradioprom's own designs similarly dominated ADP.

The most significant second-generation Soviet computer was the BESM-6. This strictly indigenous machine was designed in the first half of the 1960s by Lebedev, V. A. Mel'nikov, and their associates at IPMCE. A prototype appeared in 1965 and three more copies were built in 1967. The BESM-6 operated at 10 megahertz and was rated at more than 1 MIPS. It incorporated a number of advanced architectural features for that time, e.g., instruction and data pipelining, segmented memory, multi-programming, memory protection, fast buffer registers, and paged virtual memory.

In the early 1960s, computer priorities began a major shift from scientific toward industrial and data processing applications. Lebedev's group designed a new series of computers for information processing and control applications that incorporated dual processors with common memory, multi-machine complexes with shared memory, and real-time operations. But Minradioprom looked askance at Lebedev's (and the Academy of Sciences') effort to expand into the field of commercial and industrial computing where it was dominant. A period of sharp bureaucratic infighting ensued in the mid 1960s, from which Minradioprom emerged victorious.

Minradioprom, which had no new designs to replace the MINSK and URAL machines, saw in IBM's 1965 announcement of its new System/360 an opportunity to counter what it perceived to be Lebedev's threat to its preeminence in ADP. In the second half of the decade, Minradioprom secured top-level political support for its RIAD project which aimed to "reverse engineer" the IBM System/360 family of upwardly compatible computers. The late 1960s also saw another industrial ministry, the Ministry of Instrument Making, Automation Equipment, and Control Systems (Minpribor), begin a determined effort to occupy a larger piece of the computer field. With its ASVT systems, modeled after the PDP-8, Minpribor began a policy of emulating Digital Equipment Corporation's technology in much the same manner that Minradioprom was emulating IBM's.

These were difficult times for the Academy of Sciences. In the early Brezhnev years, the Academy was stripped of its central role as chief computer designer and was forced to give up many of its research and production facilities. By the end of the decade, Minradioprom and Minpribor had consolidated their positions not only as the nation's computer manufacturers but also as the leading centers of computer design. Not surprisingly, perhaps, they showed an increasing preference for their own designs relative to those emanating from the Academy of Sciences.

Although the Academy of Sciences was forced to concede ADP to

Minradioprom and industrial process control to Minpribor, its research and development in scientific computing continued. Lebedev's group began work on a successor to the BESM-6, a high-capacity scientific computer capable of as much as 100 MIPS. Efforts to design a machine, called the EL'BRUS, were under way by the early 1970s.

In 1965, the Council of Ministers established the State Committee for Science and Technology (GKNT) as a central coordinator of technological development policy for the entire country. In practice, the GKNT proved weak in comparison to the powerful industrial ministries and was restricted to monitoring plan assignments, although statutes allowed it a much more active role. During the 1970s, the Academy of Sciences was starved for resources and effectively removed as a major player. In the wake of the GKNT's failure to manage and promote technological development and of the Academy's emasculation, the power and responsibility for computer design fell almost completely to the manufacturing ministries. Minradioprom controlled the CMEA (Council for Mutual Economic Assistance) effort to develop mainframe computers based on IBM technology, and Minpribor did likewise for CMEA's program to develop a standard line of minicomputers based on DEC designs. The Soviet computer industry was now established on the profoundly conservative course of technological followership. Without effective leadership from the GKNT, and lacking any mechanism to spur it, indigenous computer development made little progress in the 1970s.

3. Official Plans for the 1980s

The technological stagnation of the 1970s led to a widening gap between Soviet computer technology and that of the West. This growing gap, when it was finally perceived by the Soviet political leadership, laid the groundwork for a comeback by the Academy of Sciences. In late 1984, indications mounted that a new "computer plan" was being developed in the Academy under the direction of Academician E. P. Velikhov.[6] This plan became official policy as the reins of power passed from Chernenko to Gorbachev in 1985.

On January 4, 1985, *Pravda* announced that the Politburo had "considered and basically approved a state-wide program to establish and develop the production and effective utilization of computer technology and automated systems up to the year 2000." Raising economic productivity and efficiency by accelerating scientific and technical progress, particularly in machine building and electronics, was said to be the over-arching objective of this new program.

[6] Samarskii (1984), 27; Yasmann (1985), 3.

Gorbachev, reporting to the Central Committee in June, 1985, put the matter in the following words:

> Machine building plays the dominant, key role in carrying out the scientific and technological revolution.... Microelectronics, computer technology, instrument making and the entire informatics industry are the catalyst of progress. They require accelerated development.[7]

The new informatics program, which has not been publicly disseminated, called for acceleration of production, improved quality, and the introduction of new models of computer equipment.[8] Applications of informatics technology, especially computers and microprocessors, and automation were to lead to a "comprehensive intensification of the national economy." Other major provisions of the plan were the following:

- Minradioprom and Minpribor should continue development and production of mainframe and minicomputers along the RIAD and SM lines that emulate IBM, DEC, and Hewlett-Packard designs.
- Computer output should increase by 200–230% from 1986 to 1990.[9]
- Computers already in service should be used more efficiently.
- Greater attention should be paid to minicomputer and microprocessor development for applications in specialized fields such as CAD, robotics, flexible manufacturing systems (FMS), scientific research, process control, etc.
- A major effort should be launched by the Academy of Sciences as well as by the manufacturing ministries to develop high performance "supercomputers."[10]
- A new course entitled "The Fundamentals of Computer Science and Informatics" would become mandatory in the last two years of all Soviet high schools.
- A total of 1.1 million personal computers should be produced in the period, with about half directed toward education.[11]

Under Gorbachev, information technology has moved toward center stage. Both the Soviet political and scientific leadership clearly realize that the nation lags far behind world levels in the development and application of computers

[7] *Pravda*, June 12, 1985, 2.
[8] See Vinokurov and Zuev (1985).
[9] Smirnitskii (1986), 10.
[10] Marples (1985), 1—footnotes Radio Moscow, January 20, 1985.
[11] Ershov (1986), 2.

and communications technologies. They realize, further, that these technologies are the key to progress across a broad spectrum of civilian and military purposes. The decade of the 1980s has become one in which the Soviets are struggling to stop the lag from widening and to create the conditions for closing it in the decades to come.

4. Hardware Development in the 1980s

The decade of the 1980s has been one of great activity in the Soviet computer industry. So far, it has also been one of serious disappointment for Soviet computer policy makers and users. On the one hand, the Soviet civilian computer manufacturing industry has expanded both in terms of the number of its principle actors and in terms of the range of equipment produced. The results achieved, on the other hand, continue to be disappointing both quantitatively and qualitatively.

4.1 The Major Hardware Manufacturers

At the end of the 1970s, only two Soviet industrial ministries, Minradioprom and Minpribor, were in the business of manufacturing general-purpose digital computers for civilian use. Computer components were the domain of the Ministry of the Electronics Industry (Minelektronprom). Both Minradioprom and Minelektronprom were (and are) "VPK ministries," i.e., were specifically designated members of the "military-industrial complex." In addition to radar and other radio-electronic equipment, Minradioprom supplied mainframe computers to both civilian and military users. Minelektronprom supplied electronic components for civilian and military purposes and also supplied the military with a variety of "ELEKTRONIKA" general-purpose and specialized digital computing systems.

Minradioprom and Minpribor remain important suppliers of civilian computing equipment in the 1980s and have been joined by Minelektronprom which has made its ELEKTRONIKA microcomputer systems available for general civilian users. More recently, the Ministry of Communications Equipment (Minpromsviazi) and several other producers have announced smaller systems. The following discussion of Soviet computer hardware is organized by major manufacturer.

4.1.1. The Ministry of Radio Technology (Minradioprom)

Minradioprom is the veteran Soviet computer ministry. On the scene early with its URAL and MINSK series, it solidified its position as the Soviet

Union's producer of commercial data processing equipment in the 1970s with its RIAD or "ES" systems of IBM-compatible mainframe computers. Like its American model, Minradioprom was slow to embrace personal computers but has finally done so. What follows is an account of recent Minradioprom machines.

The RIAD or "Unified Series" of IBM-Compatible Mainframes. The "RIAD" family of computers is not well known outside the Soviet bloc, although probably more copies of RIAD are installed in the world than of any other mainframe computer family except for those of the IBM Corporation. Also known as the "Unified Series" (abbreviated "ES"), these machines have been the backbone of general-purpose computing east of the Elbe since the early 1970s.[12]

Essentially, the RIAD machines are functional equivalents and technological derivatives of IBM's System/360, System/370, and Model 303x computers. By 1988, three "generations" of RIAD computers had appeared and a fourth was said to be on the drawing boards. Each of these has had a generational "lifetime" of about seven years.

RIAD represented the first attempt by Minradioprom, and the Soviet Union, at wholesale technological importation in the computer field. The ministry's earlier computer families, e.g., the URAL and MINSK series, were indigenous designs. With its wholesale copying of American computer architecture, the RIAD marked a dramatic turnabout in Soviet computer policy, one that led not only to the production of a family of IBM-compatible computers but also to the attenuation of efforts to develop indigenous computer technology in the USSR. It launched the nation upon a path of technological followership that, for all practical purposes, it has trod ever since. The principal reasons for this fateful policy shift were the following:

- Economic and political decisionmakers were dissatisfied with the results of earlier indigenous efforts. The URAL and MINSK machines were slow, unreliable, inflexible, with deficient peripherals generally and pathetically poor disk drives in particular. Small main memory and slow processor speeds dictated the use of machine language rather than higher-order languages, and applications software was limited in quantity and quality.
- Bureaucratic infighting between the Academy of Sciences and Minradioprom over primary responsibility for computer development in the USSR produced a victory for the latter.
- Soviet political leaders wished the USSR and its East European partners to be more independent of Western computer suppliers.

[12] See Davis and Goodman (1978) for an account of the early RIAD computers.

- Standardization of computer designs across the entire CMEA market area seemed to promise economies of scale and specialization.
- It seemed reasonable to focus scarce scientific and engineering resources on a coherent set of objectives rather than permitting them to be diffused and largely dissipated over a multitude of smaller projects.
- It appeared that technology transfer from the West could best be accomplished by concentrating on designs that had achieved widespread acceptance among users worldwide, not least of all in East Germany.
- The large inventory of software written for IBM mainframes seemed available at little cost.

What follows is a review of the three generations of Soviet RIAD computers with emphasis on those designed and/or manufactured in the 1980s.

RIAD-1: 1970–1977. RIAD-1, the first generation of the ES computers, was designed by a group of engineers at Minradioprom's Scientific Research Center for Electronic Computer Technology (NITsEVT) in Moscow. Aleksandr Maksimovich Larionov, NITsEVT's director, and Viktor Vladimirovich Przhiialkovskii, its deputy director, were the senior members of the RIAD design team. After Larionov's death in the late 1970s, Przhiialkovskii became Director of NITsEVT and Chief Designer of the RIAD computers, positions that he continues to hold.

Seven models of RIAD-1 computer systems were planned. These and some of their *planned* specifications are displayed in Table I. As it turned out, only

TABLE I

RIAD-1 COMPUTERS AS ORIGINALLY SPECIFIED

				Input/Output Channels		
				Multiplex	Selector	
Model	Country	Operations per second (000s)	Main memory (Kb)	Rate (Kb/sec)	Number	Rate (Kb/sec)
ES-1010	Hungary	10	8	160	1	240
ES-1021	Czech.	20	16–64	35–220	2	250
ES-1020	USSR, Bul.	10–20	64–256	25	2	300
ES-1030	USSR, Pol.	60–100	128–512	40	3	800
ES-1040	GDR	320–400	128–1024	50–200	6	1200
ES-1050	USSR	500	128–1024	100–150	6	1300
ES-1060*	USSR	1300–1500	256–2048	100–150	6	1300

* Note: The ES-1060 was shifted into the RIAD-2 era.
Source: Larionov *et al.* (1973), 3.

six systems passed the requisite CMEA tests and went into series production in 1972 and 1973.

The ES-1010, ES-1020, ES-1021, ES-1030, and ES-1040 appeared in 1972. The ES-1050 appeared in 1973. These six systems made their formal debut at a Moscow exhibition during May and June of 1973. They were displayed together with some 100 peripheral and other RIAD-1 devices. Of these, three of the computers and about 40 of the peripherals were of Soviet design and manufacture.[13] Design difficulties plagued what was to have been the most powerful of the RIAD-1 computers, the Soviet ES-1060, and when this system finally surfaced in 1978 its design had been so modified that it is properly listed among the RIAD-2 computers.

RIAD-2: 1978–1983. In 1970, IBM announced the System/370. This new computer family provided upward software compatibility with the System 360 and offered new features such as cache memory and virtual storage. Minradioprom computer designers identified several trends that they considered significant for their own program. Among them were the following: the rapid development of LSI (Large Scale Integration) which provided greater functionality and speed; fast and increasingly cheap semiconductor memory; magnetic disk storage devices with very great capacity and rapid data transmission rates; very fast cache memory; new input/output devices; virtual memory; rapidly improving performance/cost ratios; new or greater capabilities in the areas of large data base processing, multiprocessing, and teleprocessing; and architectural continuity designed to protect investments in existing software.[14]

As the RIAD-1 computers were close approximations to the IBM System/360, so the RIAD-2 systems closely resembled machines of the System/370. Eleven RIAD-2 computers are listed in Table II.[15] Included also are two "carryovers" from the RIAD-1 era. These were the ES-1033 and the ES-1060 whose gestation was so protracted that they were born into the next generation of RIAD computers.

From the beginning, the RIAD objective was to achieve compatibility with IBM at the level of logical architecture, software, and peripheral interfaces. As time has passed and the Soviet computer designers have accumulated experience and confidence, they have increasingly departed from IBM in matters of design and performance.

[13] Larionov (1976).

[14] This summary of points made in 1976 by A. M. Larionov, Riad's chief designer. See Larionov (1976).

[15] Soviet sources are inconsistent in placing specific computers into the three Riad "generations." The classification here generally follows the most recent Soviet source used in this study, i.e., Artamonov (1988).

TABLE II

RIAD-2 COMPUTERS

Model	Country	Operations per second (000s)	Main memory (Kb)	Input/Output Channels: Multiplex Rate (Kb/sec)	Input/Output Channels: Selector Number	Input/Output Channels: Selector Rate (Kb/sec)
ES-1012	Hungary	36	64	20		
ES-1015	Hungary	16	128			
ES-1022	Bul., USSR	80	128–512	50	2	500
ES-1025	Czech.	40–60	128–256	24	1	800
ES-1033	USSR	150–200	256–512	40	4	800
ES-1035	USSR	40–160	256–512	40	4	740
ES-1045	USSR	530–860	4096	40	4	1500
ES-1055	GDR	435	2048	40	5	1500
ES-1060	USSR	1300	256–2048	110	6	3000
ES-1061	USSR	1500	8192	110	6	1250
ES-1065	USSR	5500	16384	15	15	3000

Sources: Judy (1986); Lomov (1987); Artamonov (1988).

RIAD-3 Mainframes of the 1980s. In 1978 and 1979, IBM introduced a series of computers that were architecturally and functionally in the System/370 family but which bore new model designations. These were the 4300 Series and the 3030 Series. A significant attribute of these series is their adherence to the IBM 360–370 logical architecture whose memory addressing convention limits the amount of main memory to 16 megabytes. The RIAD-3 computers are subject to the same constraint.

Initial planning for the third generation of RIAD computers began in the mid 1970s, even before RIAD-2 was announced. In 1976 and 1977, a set of design objectives was set forth by the Council of Chief Designers of the RIAD Computers and was adopted by the Intergovernmental Commission for Computer Technology.[16]

The chief technical objectives of RIAD-3 were the following: to maintain compatibility with existing software and peripherals; to improve performance in terms of throughput capacity, input/output speed, memory capacity, and number of attachable terminals; to make greater use of LSI logic chips which, in turn, was to lead to reduced physical dimensions, lower power requirements, improved reliability, and higher speed; to employ more LSI memory

[16] See JS (1984).

chips—up to 4 Kb for high speed, special purpose storage and 64 Kb or larger for main memories; to introduce new memory technology for very large bulk storage; to make use of functionally oriented co-processors to control subsystems and peripherals, e.g., data flow, input/output; to use more problem-oriented processors and co-processors, e.g., symbolic processors, matrix processors, set processors, and other math processors; to improve RIAD designs of multiprocessor systems as well as network structures for remote and distributed data processing systems; to design and produce improved hardware and software for large data base management; and generally to promote greater flexibility and modularity in both hardware and software design.

Beyond these technical objectives for RIAD-3, Minradioprom sought the following results: improved price/performance ratios relative to RIAD-2; increased networking capability; greater use of data base management systems and other problem-oriented software; better implementation and usage of computers for improved economic payoff.

The development of RIAD-3 systems was planned to be in two stages. In the first stage, the primary focus was to be on (*i*) improving the component base of the systems by further development and use of new semiconductor technologies, (*ii*) development of specialized processors and software for them, and (*iii*) the transfer of selected operating system functions to hardware. This first stage of RIAD-3 development was explicitly defined as one of enhancing the performance of systems constructed according to RIAD-2 architecture.[17]

The new RIAD-3 systems were evolutionary improvements upon the preceding generation. RIAD-3 computers have maintained architectural, software, and peripheral compatibility with RIAD-2 while incorporating some technological improvements in microelectronics and design. In the period 1983–1988, production began on more than a dozen new RIAD computers, half of them being of Soviet manufacture. Those Soviet systems are listed in Table III.

The ES-1007, introduced in 1988, is the first in a new class of small RIAD computers intended for stand-alone operation or as terminals in distributed data-processing systems. Individual copies of the machine were being displayed in mid 1988 but serial production appeared not yet to have begun. Though relatively small and probably intended to be roughly equivalent to IBM's PC/370, the ES-1007 physically was more the size of an IBM System 34.

The ES-1036 is a new Minradioprom computer in the line of the ES-1020, ES-1022, and ES-1035. It offers virtual storage and dynamic microprogramming said to provide for application-oriented tailoring of system architecture to support user programs. It supports virtual machine system (VMS)

[17] See JS (1984).

TABLE III

SELECTED CHARACTERISTICS OF SOVIET RIAD-3 COMPUTERS

Model	ES-1007	ES-1036	ES-1046	ES-1061	ES-1065	ES-1066
Generation	RIAD-3	RIAD-3	RIAD-3	RIAD-2	RIAD-3	RIAD-3
Year of Appearance	1987–88	1984	1986?	1983	1985?	1987–88
In Serial Production?	Doubtful	Yes	Presumably	Yes	Presumably	Possibly
Main Processor						
Operating speed (k ops. sec.)	100	400	750, 300	1500	1600, 2000	5000, 200
Selected performance times (μsec)						
Fixed point add		0.9	0.6			0.16
Floating point add/sub.		4.81	1.69			0.32
Floating point multiply		10.1	4.06			4.6
Fixed point divide		14.3	3.8			1.6
Number of instructions		220	183	183	183	
Special & Auxilliary Processors*	S	FL, S	M, S		S	M, T, MK, S
Primary Memory						
Capacity (Mbytes)	1	2–4	8	8	16	8–16
Cycle time (μsec)			.7	.7	.85	0.68
Length of accessed word (bytes)		8–128	8–128	8–128	8–128	8–128
I/O Channels						
Maximum channel capacity, (kbyte/sec.)				10,500	30,000	18,000
Multiplexor channels						
Maximum number	1	1	2	2	4	12 universal
Data rate (kbyte/sec.)		50	160	426		
Selector channels						
Maximum number	1	4	4	6	15	12 universal
Data rate (kbyte/sec.)		1,500	3,000	1,250	3,000	
Typical Operating System		OS ES	OS ES	OS ES 6.0	OS ES 6.0	OS7 ES
Modes of Operating**	B, VM	B, MP, VS, VM, RT	B, MP, VM, RT	B, MP, VM, RT	B, MP, VM, RT	B, MP, VS, VM, RT
Programming Languages***	AS, F, PL1, C	AS, F, F4, PL1, C, RPG	AS, F4, A, PL1, C, RPG	ASMG, F, C, PL1, RPG	ASMG, F, C, PL1, RPG	AS, F, PL1, C, A

Keys to Abbreviations:

* FL—Floating point; S—Service; M—Matrix; T—Text; MK—Macro Conveyor

** B—Batch; MP—Multiprogramming; VS—Virtual storage; VM—Virtual machine; RT—Real time

*** AS—ASSEMBLER; F—FORTRAN; F4—FORTRAN IV; PL1—PL/1; C—COBOL; RPG—RPG; A—ALGOL

Sources: Artamonov (1988); Elektronno (1988); Judy (1986); Kezling (1986).

operations under different operating systems. It, like other larger RIAD-3 machines, is available in multi-machine configurations. The machine went into serial production at the Minsk computer plant in 1984.[18]

The ES-1046 is the latest in a line of Minradioprom machines that previously have included the ES-1030, ES-1033, and ES-1045 computers. Like its predecessors, it was designed by the Ministry's Erevan Scientific Research Institute of Mathematical Machines. It passed its international RIAD tests in 1984 and serial production had begun by 1985 at the Kazan computer plant. The chief designer was A. Kuchukian, a Lenin Prize laureate.[19] The ES-1046 is said to have greatly improved diagnostic and fault location capabilities that can locate 99% of all failures to within two or three exchangeable components. A full system checkout requires five minutes or less. The ES-1046 is said to be available in 11 basic configurations including dual-processor and dual-machine variants. It employs a service processor, a matrix processor, and a machine graphics device. The performance/price ratio for the ES-1046 is said to be twice as favorable as that for the ES-1045, its immediate predecessor in the Armenian subfamily of Minradioprom computers.

The ES-1061 is the successor to the ES-1060 computer. Although Artamonov (1988) designates it a RIAD-2 machine, it is discussed here because it appeared first in this decade, contemporaneously with the RIAD-3 generation. Design work is said to have begun in about 1980 by a team from Minradioprom's Moscow NITsEVT and its Minsk Ordzhonikidze Computer Association. The design team was headed by V. V. Przhiialkovskii, the General Designer of the RIAD computers. Serial production began in 1983. It was to be offered for export in 1984.[20] Reliability is said to be 150% of that of the ES-1060.

The ES-1065 is a high-performance Soviet computer that, like several other top of the line Soviet systems such as the ES-1050 and ES-1060, has been a long time aborning. The machine was originally intended to enter production by 1977 as the most powerful of the RIAD-2 computers. In early 1984, the ES-1065 was still in prototype and the Minsk computer factory was planning to begin production only in that year.[21] The original specifications called

[18] Information on the ES-1036 is from Artamonov (1988), 192–195; Dujnic and Fundarek (1983); Loeschner and Kasper (1984); Zamorin *et al.* (1984); and *Sovetskaia Belorussiia* (1984).

[19] Information on the ES-1046 is from Artamonov (1988), 192–195; Dujnic and Fundarek (1983); Zamorin *et al.* (1984); Kuchukian *et al.* (1985); Selivanov (1987); *Sovetskaia Litva* (1985); and Musaelian (1985).

[20] Information about the ES-1061 is from Artamonov (1988), 192–195; Dujnic and Fundarek (1983); *Szamitastechnika* (1984), *Sovetskaia Belorussiia*, April 3, 1983, 1 and January 8, 1984, 1; and *Elorg Informiruet* (1983).

[21] Larionov (1977).

for it to be a 4.5 MIPS machine. Later information indicates that its speed is no greater than 2 MIPS.[22]

The ES-1066 is another machine designed by the Armenian group at Minradioprom's Scientific Research Institute of Mathematical Machines in Erevan.[23] Its maximum speed is rated at 12.5 MIPS, and it is said to perform 2 MIPS in data processing tasks and up to 5.5 MIPS in scientific computing. The ES-1066 is driven by the ES-2366 CPU, which is highly buffered and pipelined with five-level interleaving. With a cycle time of 80 nanoseconds, this is the fastest single processor that the Soviets had revealed by mid 1988. It makes extensive use of specialized microprocessors and microprogram control units. Input and output, for example, are controlled by the ES-2666 I/O processor which supports up to 20 megabytes per second of data flow.

The ES-1066 is the first RIAD to be equipped with the ES-5080 disk drives that offer reliability said to be much improved in comparison to previous Soviet RIAD mainframes. From its specifications, the ES-1066 appears to be a high-performance machine. Overall system performance is likely to be limited by the shortage or unavailability of large-capacity disk drives and other peripheral devices. Final judgement on its quality and operational performance must be suspended until user reports become available. As with most larger RIAD models, serial production of the ES-1066 posed production difficulties. Volume production was underway in 1987, about two years behind schedule.

RIAD-4. In late 1984, preliminary development work on the next generation of RIAD computers was said to be in progress. Some of the goals of this development effort were stated to be the following: maintenance of software compatibility with previous RIAD systems; more advanced architecture permitting greater expansion and efficiency; faster processor operating speeds; greater real main storage capacity of the CPUs; greater external storage capacity and faster data rates; improved ability to build local and extended networks; improved I/O devices; greater user friendliness; improved operating systems that will increase throughput as well as provide more features and functions; and better diagnostics, reliability, and maintainability.[24]

The next generation of RIAD computers is said to incorporate a new multiprocessor architecture based on problem-oriented and functionally

[22] Information about the ES-1065 is from Artamonov (1988), 192–195; Novak (1983).

[23] Information about the ES-1066 is from Artamonov (1988), 192–195; Lomov (1987); Selivanov (1987); Zamorin *et al.* (1984); and *Sovetskaia Litva* (1985).

[24] Jungnickel (1984). The author, Dr. Hang-Georg Jungnickel, was Chief Designer Engineer for ES computers at the Robotron combine, which is the manufacturer of Riad systems in the GDR.

oriented processors. The problem-oriented processors are for matrix operations, symbolic processing, and support of problem-oriented languages. The functionally oriented processors include input/output processors, telecommunication and set processors (which are to optimize processing of sets on external memories). The individual processors are to be connected by a high-speed bus.

Other aspects of the new systems are said to include large-capacity main memories (potentially up to 2048 megabytes) built of 64 Kb and larger memory chips. The various specialized processors will be served by dedicated storage of up to 256 kilobytes. External memory is to be controlled by independent control processors which will also serve as virtual storage control. Mention is made of external memory devices consisting of several 100-megabyte units on cylindrical magnetic "layers" that may be a variety of drum storage. New operating systems, compatible with existing ones, are said to be under development to support the transition to the multiprocessor systems.

Without more information, it is difficult to divine the specific features of the RIAD-4 generation of computers. From the hints provided, however, it appears likely that new systems will display many characteristics of the IBM 308x Series. The references to "a more advanced architecture," "multiprocessor systems," and "greater real main storage capacity" point strongly in that direction.

Minradioprom computer designers and their East European colleagues have accumulated nearly 20 years of experience in the design, development, and production of RIAD computers. That experience inevitably has developed a degree of expertise that was absent at the beginning. Increasingly, the RIAD designers are departing from a strict adherence to IBM designs.

While preserving upward software compatibility with previous RIAD (and IBM) systems, and remaining within the IBM "mainstream," Minradioprom is emulating other IBM-compatible manufacturers such as Amdahl, NAS, Fujitsu, and Hitachi. In other words, they are trying to "add value" to the basic IBM design, where "value" is to be understood in terms of the ministry's perceived needs and priorities.

The question remains as to when the next set of RIAD computers can be expected to arise from the drawing board. Since 1972, a new group of RIAD computers has appeared at approximately seven-year intervals. Thus, the interval between RIAD-1 (1972–1973) and the RIAD-2 (1978–1981) computers was six to eight years. Likewise, the interval between the RIAD-2s and the RIAD-3s (1984–1988) was six or seven years. A seven to eleven year technological lag of RIAD behind IBM has also been observed.

If the previous pattern of time lags behind IBM still held, more powerful RIAD-3 computers, which could resemble the IBM 3080 Series, should have

appeared by 1986 and 3090-like RIAD-4s should appear by about 1990. The fact that RIAD-3 computers comparable to the IBM 3030 models entered serial production only in the 1984 to 1988 timeframe suggests that Minradioprom is experiencing greater difficulty with the higher performance systems. It now seems likely that the RIAD-4 machines will be delayed until the 13th Five Year Plan, i.e., sometime in the early 1990s.[25]

One hint of Minradioprom plans was given in 1983 by a senior official who spoke of a future RIAD computer dubbed the ES-1087.[26] This machine, with a MIPS rated two and one half times greater than that of the ES-1065, was to be in production by 1990. Such a machine would have a capacity not too dissimilar from that of an IBM 3081D which appeared in 1981.

Another tidbit of information was provided in an early 1988 reference to the ES-1068. This new Soviet "computing complex" is said to be capable of 600 MOPS when equipped with "special matrix processors designed to accomplish specific tasks." The principle ES-1068 architecture is said to be well known to world computer science but the Soviet implementation is claimed as a first. This, plus the indication that the ES-1068 is intended for "such complex tasks as geological prospecting for major mineral deposits and constructing models of ecological processes," leads us to conjecture that the machine employs a massively parallel architecture of the MIMD (Multiple Instruction, Multiple Data) type.[27] The ES-1068 may be Minradioprom's response to Minpribor's rather successful PS-x000 series of massively parallel computers which also are used for geological work.

A summary chart of operational characteristics intended for future RIADs is presented in Table IV.

Toward a Fifth Generation RIAD. A few clues to Minradioprom's thinking about a RIAD-5 have emerged.[28] The first premise appears to be that program compatibility with previous RIAD software continues to be a *sine qua non.* The second is that improvements in reliability, service, usability, and peripheral assortment are more important to Soviet users than the gains that might be realized from greater architectural sophistication. That, of course, does not preclude the use of better and less costly microcircuitry, microprogramming, specialized processors, etc.

[25] This conjecture is lent credence by a Hungarian source in which the expectation is expressed that the ES-1034, a Riad-3 computer, will remain in supply throughout the 1986–1990 period. See Nanassy (1985).

[26] Novak (1983). The author is quoting M. E. Rakovskii, the Director of CMEA's Intergovernmental Committee for Computer Technology.

[27] See Marchuk (1986), 99–108.

[28] In an article by the RIAD Chief Designer, Przhiialkovskii (1987).

TABLE IV

BASIC PERFORMANCE TARGETS OF FUTURE RIAD COMPUTERS.

Characteristic	Time period 1985–90	1990–95
Degree of IC integration, (number of logical elements)	500–1000	5000
Number of ICs in CPU	600	50
Processor speed, MIPS	10	100
Machine cycle time, ns	30–50	3–5
Main memory chip size, Kbits	64	512
Fetch time from main memory, ns	150	20
Fast buffer memory size, Kbytes	64	1024
Fetch time from buffer memory, ms	18	3–5

Source: Maliarskii and Terekhov (1987).

A major thrust of future RIAD development is toward better user interfaces, an interesting direction since "user friendliness" has never been a strong point of Soviet mainframes. A second thrust is said to be toward integrated networks combining systems from PCs to supercomputers. It appears, in short, that Przhiialkovskii and his fellow Minradioprom designers continue their inclination to follow Western and Japanese leads in creating future generations of RIAD computers.

Measuring the IBM vs. RIAD-3 Lag. From the beginning, RIAD designers have followed the lead of IBM. It is fitting to provide a general assessment of RIAD's performance relative to that of IBM.

With the introduction of its 308x Series computers in 1981, IBM moved to a new architecture which permits vastly greater main memory as well as more processing power.[29] No Soviet computer with 3080 Series (much less 3090 Series) architecture was known to have been announced or shipped by mid 1988. The ES-1066 is the first Minradioprom RIAD computer to display characteristics of the IBM 3080 Series.

[29] The address portion of the instruction format common to the IBM 360-370-3030-4300 computers is 24 bits in size. Such an instruction can address a memory location up to 16,777,216 which is 16 megabytes. The IBM 3080 Series of computers employs what IBM calls its Extended Architecture (XA) with 32 bit addressing. With this, the computer can address locations up to 2,147,483,648 which is 2 gigabytes. Another feature of the 3080 and 3090 series of IBM computers is their use of multiprocessors packaged as integral central processing units.

TABLE V

MATCHING SOVIET RIAD COMPUTERS WITH IBM COUNTERPARTS

Model	Country	Year shipped	Closest IBM counterpart	Year IBM shipped	RIAD Lag in years
RIAD-1					
ES-1020	USSR	1972	IBM 360/30	1965	7
ES-1030	USSR	1972	IBM 360/30	1965	7
ES-1050	USSR	1973	IBM 360/65	1965	8
RIAD-2					
ES-1022	USSR	1975	IBM 360/44	1966	9
ES-1033	USSR	1976	IBM 360/50	1965	11
ES-1035	USSR	1977	IBM 370/135	1972	5
ES-1045	USSR	1982?	IBM 370/148	1977	5
ES-1060	USSR	1977	IBM 370/165	1971	6
ES-1061	USSR	1984	IBM 370/168	1973	11
RIAD-3					
ES-1036	USSR	1984	IBM 370/138	1976	8
ES-1046	USSR	1986?	IBM 4341?	1979	7
ES-1065	USSR	1985?	IBM 3033N?	1980	5
ES-1066	USSR	1987	IBM 3033U?	1978	9

Sources: Artamonov (1988); *Data Decisions* (1983); Judy (1986); Phister (1979).

Unlike the RIAD-1 machines, the RIAD-2 and RIAD-3 computers are not close clones of IBM originals. One-to-one matching is not possible with the later models. We have tried, nevertheless, to make some comparisons and Table V shows a matching of recent Minradioprom RIAD computers with IBM machines displaying similar architectural and performance characteristics.

By definition, the imitator lags behind the imitatee. The discovery, therefore, that RIAD lagged temporarily behind IBM in producing computers with comparable CPU and memory capabilities provokes small surprise. The interesting question is: How great is the Minradioprom mainframe lag and how has it changed with the passage of time?

Soviet RIAD-1 computers lagged behind their IBM System/360 counterparts by seven or eight years; the average lag was 7.3 years. RIAD-2 computers lagged behind similar IBM System/370 systems by from five to eleven years with an average of 9.4 years. The four Soviet RIAD-3 computers so far released have appeared from five to nine years after the IBM systems with which they have been matched in this study; the average RIAD-3 lag has been 7.25 years.

Minradioprom's RIAD project is an effort to produce IBM-compatible computers. How does the RIAD effort compare to Western and Japanese plug compatible manufacturers, the so-called "PCMs"?[30]

Taken as a group, the Western and Japanese PCMs offer an impressive array of computers. In the mid 1980s, the PCMs were offering more than 54 computer models that were compatible with the IBM System/370 and the 4300, 3030, and 3080 Series machines. This compared with 21 offered by IBM itself and 11 RIAD computers announced by all the CMEA countries taken together.

As a group, the PCMs have lagged very little behind IBM in bringing their products to market. In contrast to an average RIAD-3 lag behind IBM of more than seven years, the PCMs lagged an average of only 0.44 years behind IBM. Furthermore, the alacrity with which the PCMs have brought forth their clones has increased with time. Only a few PCM versions of System/370 machines were still for sale in 1985 but their average lag behind IBM was 4.33 years. The average lag for the 303x look-alikes was only 0.73 years. For the 4200 Series, it was 0.23 years, and for the 308x, the PCM competition actually beat IBM to market by an average of 0.05 years.

Not only were the PCMs able virtually to eliminate the time lag behind IBM, their computers were, on the average, 0.48 MIPS more powerful than the IBM computers with which they were designed to compete. Some examples illustrate the point. Amdahl/Fujitsu's "imitation" of the IBM 3032 not only beat IBM to market by a year but was 1.5 MIPS more powerful when it got there. NAS/Hitachi's product arrived two years earlier and 1.25 MIPS more powerful than the IBM 3083J. The PCMs led IBM also in a number of important areas of technology, e.g., in the early use of very large scale integrated (VLSI) circuitry.

The major PCMs have maintained a blistering pace of new product introduction and technological innovation. In many cases, they have undoubtedly forced IBM to bring new computers to market earlier than it would have preferred. Compared with this example of competitive markets at work, the performance of CMEA's RIAD effort is unimpressive. The RIAD-3 average lag of over seven years behind IBM was greater than the average lag of any of the major PCMs. As a manufacturer of plug compatible computers, the Soviet-East European RIAD consortium brings up the rear.

That the RIAD lag behind IBM has remained essentially invariant during the course of nearly two decades is rather surprising. Several factors might have caused the Soviets to pick up the pace. For example, information about Western computer technology in general and IBM's designs and intentions in

[30] Prominent among the PCMs are Amdahl, National Advanced Systems, Nixdorf, Hitachi, and Fujitsu. For more on the PCMs' performance, see Judy (1986), Appendix B.

particular surely must come faster to RIAD engineers as the Soviet industrial intelligence apparatus has matured.[31] RIAD engineers have accumulated 20 years of experience in building IBM-compatible computers. Has their expertise not increased correspondingly?

It would seem that, while the engineering and design sides probably have improved over the years, the industrial basis of computer production has failed to improve correspondingly. The ponderous Soviet planning and industrial establishment has moved too slowly to support high-tech manufacturing. Many of the old problems of the economic system remain or even worsened with the passage of time; among them are those of fractured responsibility for R&D and production, monopolistic ministries, organizational infighting, bureaucratic bumbling, inappropriate success indicators, disincentives for innovation, inattention to quality control standards, and sluggish industrial supply.

Evaluating the RIAD Performance. In the space of two decades, the Soviet Minradioprom and its CMEA partners have created the capability to design and manufacture IBM compatible mainframes. Starting from a very modest technological base, the RIAD consortium by 1985 had brought three generations of computers to market.

Every RIAD generation has improved substantially on its predecessor. This is true of both individual models and of the family as a whole. While the RIADs have not gained technologically on either IBM or the PCMs, Minradioprom computer engineers have proven themselves capable of designing powerful mainframe computers. Furthermore, the RIAD designers have apparently accomplished many of the objectives that were set before them at each generation. Despite considerable improvement over the years, however, the Soviet RIAD computers appear not to have developed strong loyalties among their users. Indeed, even Soviet users prefer RIADs manufactured in East Germany to the domestic products.

Much of the users' dissatisfaction with the Soviet RIADs can be traced to Minradioprom and its suppliers, particularly Minelektronprom. Soviet manufacturing weaknesses have adversely affected the design and production of basic components and of peripherals, particularly of disk storage devices. Minradioprom still uses 64 Kb memory chips, for example, at a time when 1 Mb chips are in widespread use in the United States and Japan. The RIAD computers continue to employ old bit-slice processor chips. Until the early 1980s, RIAD central processors were using Minelektronprom's K589 series

[31] The "industrial intelligence apparatus" is understood here to include everything from espionage, to imports in defiance of COCOM restrictions, to the processing of unclassified western technical literature.

chip and now they use the K1800 series. These and other families of Soviet chips are described below in Section 4.3.

Other persisting problems of the RIAD mainframes have been unsatisfactory reliability, inappropriate configurations for user purposes, shortages of spare parts and supplies, software inadequacies, generally poor levels of user support, high cost, and insufficient levels of output.

These problems, which are not confined to Minradioprom, must be solved before the Soviet Union can be said to have developed a satisfactory mainframe computer industry.

Personal Computers from Minradioprom.

Like IBM, Minradioprom was slow to climb aboard the PC bandwagon. Admittedly, that bandwagon began to roll much later in the Soviet Union than in the United States and Minradioprom may have had to contend with certain negative official attitudes toward PCs until 1985. But it also seems likely that Minradioprom engineers and managers were inveterate "mainframers" just as were many engineers and managers in large American computer companies before Philip Estridge sprung the product of his PC "skunk works" in Boca Raton upon the computer world. In any case, Minradioprom waited until 1982 to begin designing personal computers.

AGAT, An Apple-II Clone.

The AGAT was the Soviet Union's first personal computer and, true to Minradioprom tradition, was the clone of an American original, in this case the Apple-II.[32] Since the AGAT was intended primarily for schools, it was not unreasonable that Apple was taken as the model since more educational software was available for the Apple family than for any other brand of personal computers.[33] Design work on the AGAT began in 1982 and early models were in use by 1983. The machine entered serial production at the Lianozovskii Electromechanical Factory in 1984.

Whereas Jobs and Wozniak were able to use the "off the shelf" Motorola 6502 chip for the Apple-II, Minradioprom found itself constrained to build up the AGAT's 8-bit CPU using Minelektronprom's K588 CMOS bit-sliced processor. This, combined with slow disc access, are probably what make the AGAT run up to 30% slower than the Apple-II.[34] In addition, the machine provided CP/M compatibility via an Intel 8080-compatible Minelek-

[32] Information on the AGAT comes primarily from Artamonov (1988), 207; Ioffe (1984); Savel'ev (1987a), 120–124; *Informatika i Obrazovanie*, 1987: 6, inside cover; and from the senior author's personal observations.

[33] Ershov (1987).

[34] For some run-time comparisons performed by an American visitor to the USSR, see Bores (1984), 135, *passim*.

tronprom K580 coprocessor. The AGAT was configured with 42 Kb of ROM and 64 Kb (expandable to 256 Kb) of RAM. External storage could be on one or two 258 Kb floppy disk drives (ES-5088 or ES-5089) or audio tape cassettes. The RGB TV monitor provided either 40 × 24 or 80 × 24 character display as well as three bit-mapped color graphics modes; an audio generator supplied sound. Input-output was by two programmable parallel and one RS-232C serial ports.

AGAT DOS, the operating system provided by Minradioprom, was strictly analogous to Apple-DOS as was most of the rest of the machine's basic software endowment. BASIC-AGAT was the counterpart of Apple BASIC. A version of Applesoft apparently was housed in ROM. The AGAT's text editor, file manager, and graphics editor all had their Apple counterparts.

Fortunately for students using the AGAT, another line of software for the machine was developed by the Computing Center of the Siberian Division of the USSR Academy of Sciences. This line, called "SCHOOLGIRL" (*SHKOL'NITSA*), included a much-improved DOS, a LOGO-like programming language called "RAPIR," and a graphics package called "SHPAGA."

The AGAT's advantage over other Soviet PCs for educational purposes is said to lie in its color graphics capability that makes possible interesting instructional software. According to Soviet users, the AGAT's main disadvantage lies not in its sluggishness but, rather, in its extremely poor reliability. Its floppy disk drives and keyboard are said to be particularly prone to failure. According to one user, drive failures occur with "catastrophic frequency and often with irreversible consequences." Disks that read on one drive may be unreadable on another. Machines are constantly in for repair and, even so, minor failures (e.g., individual keys inoperable, one color unavailable) must simply be overlooked.[35]

Because of its dubious quality, the AGAT has been the butt of much serious criticism from Soviet officials as well as computer users. Rumors and reports of its death, i.e, the discontinuation of its production, have been recurrent. The most recent of these was a report that a state commission had decided to withdraw the AGAT from production in early 1987 in order to replace it with the KORVET.[36] But the AGAT seems unwilling to die.[37] Hundreds, perhaps thousands, of them are installed in Soviet schools. Some of the Soviet Union's best educational software operates best on the AGAT and, of course, the international library of Apple educational software continues to grow. It would not be surprising, therefore, to see an improved version of the AGAT

[35] Basin (1988); Yasmann, (1987).

[36] *Molodezh' Estonii*, June 3, 1987, as cited in Yasmann (1987).

[37] A journal article published early in 1988 indicated that the AGAT was still in serial production. See Petrov (1988).

make its appearance even though it has some formidable enemies in Soviet official computerdom.

ES-184x, **IBM PC/XT Clones** With the ES-1840 and ES-1841, Minradioprom returned to the familiar ground of copying IBM originals.[38] The "ES" model designation was rather surprising since it had always before been reserved for RIAD mainframes. The explanation appears to lie in the fact that Minradioprom, following the IBM lead, is touting these PC clones as professional workstations that can be connected to the ministry's RIAD mainframes. Ministry officials also may harbor hopes that their designs can be made the CMEA standard. If that were to occur, it would make a travesty of CMEA computer collaboration since many better PC clones are made by other countries in Eastern Europe and also by other ministries in the USSR.

The ES-184x machines are manufactured by Minradioprom's Minsk Computer Works and are less-than-perfect copies of the IBM originals. They differ, first of all, in that their motherboard layout differs substantially from that of the IBM machines. PC-compatible graphics and other boards will not fit the ES-184x expansion slots. No PC communications packages will work on them because their I/O port is not RS-232-C compatible.

The ES-184x machines differ also in that they do not use the Intel 8088 or its clone for its CPU. Rather, they employ the Soviet K1810VM86 chip which is a 4 megahertz Minelektronprom copy of Intel's 8086 microprocessor. The machines are configured with 256 Kb (expandable to 640 Kb) of RAM, a Cyrillic and Latin character keyboard with 92 keys including 10 that are programmable, a 80 × 25 monochromatic display, and "quasi" RS-232 and Centronics ports.

The ES-1840, announced in 1986, offers two 320 Kb floppy disk drives and differs from the IBM PC in that the drives are bulkier and are housed separately from the main unit. Furthermore, the disks are not perfectly compatible with the IBM format. The ES-1841, which appeared in 1987, is an IBM-PC/XT semi-compatible machine equipped with a Bulgarian 10 megabyte Winchester disk. A mouse is available but is implemented differently than on the IBM-PC and many programs intended for the latter do not work on the ES-1841. Other peripherals include a color monitor and a plotter. Although one of the present authors saw several ES-1840 systems on display and in operation during his travels in the USSR during the summer of 1988, he never saw a ES-1041 or any of the other peripherals described. The systems displayed normally were equipped with Epson or Robotron (East German) printers.

[38] Information on the ES-1840 and ES-1841 comes from Pykhtin (1986) and marketing brochures supplied by Minradioprom, and personal observations of the senior author.

Like the machine itself, the ES-184x software supplied by Minradioprom is copied or closely derived from American originals. The standard ES-184x operating system is M86, a Soviet version of Digital Research's CP/M-86, which, according to Minradioprom, will support WORDSTAR, SUPERCALC, D-BASE II and III, SYMPHONY, and other popular American software packages. Also available is ALFA-DOS, a Soviet version of MS-DOS v. 3.2. Programming languages include ASM86 (a Soviet version of Digital Research's assembly language), as well as BASIC M86 and PASCAL M86 which are Soviet versions of those popular languages. Applications software packages for the ES-184x include ABAK (a Soviet version of the SUPERCALC spreadsheet), SLOG (a WYSIWYG Russian and Latin character version of the WORDSTAR wordprocessor), and DELOGRAF (a business graphics package probably patterned after an American original).

Users of the ES-184x dispute Minradioprom's claims for software compatibility with the IBM-PC. They point out that the hardware differences make the ES-184x incompatible not only with true IBM compatibles but also with other Soviet "PC compatibles" like Minpribor's ISKRA-1030 and Minpromsviazi's NEIRON series.[39]

PK-80xx; CP/M Compatible PCs. The PK-8001, PK-8010 and the PK-8020 are members of a new Minradioprom family of eight-bit computers based on Minelektronprom's KR580VM80A microprocessor, a 2.5 megahertz imitation of Intel's 8080A chip.[40] MikroDOS, a Soviet version of CP/M-80 is the standard operating system for the PK-80xx machines.

The PK-8001 is configured as a small personal computer with main memory of 16 to 64 kilobytes. Standard external memory is on audio cassette although provision is also made for 8″, 5.25″, and 3″ disk drives. Soviet authors compare this machine to Radio Shack's TRS-80 and put its speed at 625 thousand operations (register-to-register) per second or about 25% slower than the IBM PC/XT.

The PK-8010 is normally equipped with 64 Kb of RAM, 24 Kb of ROM, 48 Kb of dedicated graphics memory, black and white monitor (512 × 256 pixels) and is intended primarily as a student's workstation in a KORVET classroom network. The PK-2020 may be configured with either monochromatic and/or color monitor, one or two 800 Kb floppy disk drives, dot matrix printer, and an audio cassette tape storage device. It is intended to serve

[39] Shirokov (1988).

[40] Information about the PK-8001 is from Velikhov *et al.* (1986). Information on the PK-8010 and PK-8020 is from Sulim *et al.* (1986), 74; Velikhov (1987a), 28; Driga (1986), 66–68; and *Informatika i Obrazovanie*, (1987) (2), *passim.*

as the teacher's workstation in a KORVET classroom network. Main memory is said to be expandable to 256 Kb. The detachable keyboard, which accepts input in both Cyrillic and Latin characters, is augmented by five programmable function keys and a numeric keypad that doubles as cursor control.

KORVET: A Classroom Network. The KORVET is a classroom configuration of up to twelve PK-8010 student computer workstations networked together with one PK-8020 teacher's workstation.[41] In network mode, the KORVET's operating system presumably operates a Soviet modification of MP/M-80. Standard programming languages are said to include a Soviet version of BASIC compatible with Microsoft's MSX BASIC, PASCAL, and RAPIR.

On the face of it, an 8-bit CP/M machine would seem an unlikely choice for the Soviets as one of their main educational computers. In the past, the Soviets have placed heavy weight on the quantity and quality of software that they could "borrow" when they were deciding which American computer designs to emulate. But in this case, very little Western educational software will run under CP/M. The Apple II, Commodore 64, and TRS-80, all with proprietary operating systems, were the 8-bit computers of choice for American schools in the early 1980s. In the pre-IBM PC era, the CP/M machines held sway only in business applications. Even in that field, they were quickly eclipsed by PC-DOS/MS-DOS machines after the IBM PC was announced in 1981. Why, then, the choice of 8-bit CP/M machines for Soviet schools?

The history and tribulations of the PS-80xx and the KORVET classroom network cast light on how that decision was made and also starkly illustrate some of the fundamental problems of the Soviet computer industry.[42] The computer was conceived in 1985 in a laboratory of Moscow State University's (MSU) Institute of Nuclear Physics. Lacking an appropriate computer for their experimental work in low-temperature plasma physics, Professor Alexander T. Rakhimov and a young associate, Nikolai Roi, designed and built their own machine. As it happened, their "dean" at MSU was the head of the Department of Physics and Plasma Physics who was none other than Academician Evgenii Pavlovich Velikhov, soon to become Vice President of the Academy of Sciences and head of its Department of Informatics, Computer Technology and Automation as well as chief scientific advisor to Mikhail Gorbachev.

Velikhov was sufficiently impressed with his colleagues' handiwork to

[41] Information for the KORVET comes from the same sources as that for the PK-80xx machines described earlier.

[42] This KORVET case is based on the senior author's interviews and Grif (1988).

convene a meeting of computer specialists and industrialists at the Presidium of the Academy of Sciences for the purpose of demonstrating the new computer. At that meeting, the unanimous opinion was that the KORVET should be mass produced. Another demonstration was scheduled, this one at the Council of Ministers building, to be attended by top industrial leaders including some ministers. Again, the response was enthusiastic and a decree went forth from the Central Committee of the Communist Party and the Council of Ministers calling for mass production of the machine which was, by this time, named the KORVET. Production was planned to be in the following numbers.

Year	Planned Production
1987	10,000
1988	36,000
1989	84,000
1990	120,000
1992	250,000

The responsibility for preparing final documentation and prototypes was assigned to the Scientific Research Institute of Calculating Machines ("NIISchetmash") while Minradioprom's Baku factory "Radiostroenie" was designated the producer. NIISchetmash said that it would take three (sic!) years to complete the working documentation, an interesting indication of the normal pace of design work in the Soviet computer industry. In the end, however, they were able to complete the design and produce a prototype school computer laboratory in one year. The KORVET was approved by a state certification board in January, 1986 and its mass production was recommended. The committee stated that the KORVET design satisfied the requirements for educational computing and was technically superior to other Soviet computers designed for this purpose.

At about the same time, a competing design for a school computer came before the state certification board for approval. This machine, the UKNTs, was to be produced by Minelektronprom and initially was not approved by the certification board. More than 10 specialized integrated circuits were required for the UKNTs, and the KORVET designers charge that Minelektronprom, which is the Soviet Union's monopoly producer of integrated circuits and other electronic componentry, gave total priority to its "own baby" and failed to meet commitments to Minradioprom.

According to Professor Rakhimov, a top Minelektronprom official explicitly invited the MSU designers to abandon Minradioprom and join forces

with his own ministry. When Rakhimov refused to do this, citing the existing decision to manufacture the KORVET in Baku, his Minelektronprom interlocutor cited the interministerial competition and predicted that UKNTs would live and the KORVET would die.

As it turned out, Minradioprom's Baku factory has been plagued by inadequate quantity, inappropriate assortment, and low quality of components from Minelektronprom. All parties associated with the KORVET are convinced that their effort has been victimized by Mineletronprom's favoritism toward its own design, the UKNTs.

Early hopes that serial production of the KORVET would begin early in 1987 proved to be sanguine. Minradioprom was to begin serial production in the fourth quarter of the year, but only a few systems were produced. Output in 1988 is well below target and many otherwise completed machines are said to be waiting at the factory for monitors that Minelektronprom has failed to supply.

Concern about the longer-run feasibility of mass producing the KORVET arises from the fact that the Baku Radiostroenie plant relies almost exclusively on manual labor in its production. What equipment they have is said to be mainly homemade. Entreaties to the planning authorities and Minradioprom have failed to elicit the equipment necessary to ramp production to the planned levels. The idea of purchasing a Japanese turnkey computer manufacturing factory has attracted favorable attention. A television factory in Lvov now produces a million television sets per year in such a Japanese turnkey plant. For computers, however, not only foreign exchange shortages but also COCOM (Coordinating Committee for Multilateral Export Controls) restrictions stand in the way.

The KORVET case illustrates the following weaknesses of the Soviet computer industry:

1. Computer design, especially PC design, is frequently a haphazard matter: The KORVET arose almost accidentally from a physics laboratory at MSU.
2. The three functions of initial design, working documentation and prototyping, and manufacturing are disjointed. Three totally different organizations with three different sets of objectives have been involved in the KORVET.
3. The oligopolistic structure of Soviet industry is a powerful brake on progress in computer technology. Minelektronprom's interest in and favoritism toward its own computers conflicts with its position as the USSR's single supplier of computer components.
4. No satisfactory alternative to market competition exists in the Soviet

economy to identify meritorious computer designs and to mobilize resources for their production.

5. Shortages of components and manufacturing facilities impede the expansion of Soviet computer production.

Future Directions for Minradioprom PCs. Minradioprom has provided clear indication of its dreams of becoming a major Soviet PC producer.[43] The ministry's strategic orientation is primarily toward the professional workstation market, especially where networking or other connections to mainframes are desired, and secondarily toward the education market. The truly "personal" computer user hardly enters the ministry's present plans for the future.

The IBM PC/XT semi-compatible ES-184x machines are the extent of Minradioprom's present and rather meager set of professional workstations, a set the ministry calls the "first series" (hereafter "MRPC-1") of its PC offerings. The general specifications for Minradioprom's "second series" (MRPC-2) call for a set of machines to be built around a 32-bit processor analogous to the Intel 80286 or 80386. There is no Soviet counterpart to either of these chips and none is said to be on the horizon. State acceptance tests for the first MRPC-2 machine, the ES-1842, were said to be in preparation in early 1988.[44] Minradioprom may use an East German 80 x 86 clone which is thought to be under development. Alternatively, Far Eastern 80 x 86 clones may be imported in quantity. Minradioprom plans call for the MRPC-2 to execute the RIAD instruction set either by emulation or by coprocessor. Main memory would consist of one or two megabytes of RAM. Disk storage would be on 500 Kb floppy disks and hard disks with capacities of 12.76, 25.5, and 40 megabytes.

The MRPC-2 machines are intended to support a host of operating systems including CP/M-86, MS/DOS, VM/PS, UNIX, and a synthetic system CCP/M-86 said to combine the functions of CP/M-86 and MS/DOS. Programming languages are to include those available under MS/DOS plus those supported on the RIAD mainframes, e.g., ADA, FORTH, C, and PROLOG. Application packages would include a word processor (probably related to WORDSTAR), DBASE II, and SUPERCALC.

Minradioprom's business plan calls for the MRPC-2 to sell in the range of 8–10 thousand rubles ($13,000–17,000) and for annual output to be in the "hundreds of thousands." The plan also calls for the design and production of a range of PC peripherals.

[43] This section is based on the senior author's interviews and Lopato *et al.* (1986).

[44] Shirokov (1988).

The ES-1840 and ES-1841 are clearly inferior to the IBM-PC/XT of 1981 vintage. Tomorrow's Minradioprom ES-1842 probably will be inferior to yesterday's IBM-AT. The ministry is nowhere close to designing a computer comparable to IBM's PS/2 whose Intel 80386-based Model 80, as Shirokov (1988) ruefully put it, "surpasses our most productive computer for scientific and technical computations—the BESM-6." The thought that American PCs will be using the Intel-80486 chip in the early 1990s greatly depresses Soviet computer users struggling to obtain and then use machines that are inferior to Intel 8088-based PCs.

Minradioprom's dreams of becoming a major supplier of PCs depend critically on its ability to master the techniques of designing and then mass producing high-quality, reliable, electronic consumer products, something it has not demonstrated up to now. They also depend on vital factors over which the ministry has little or no control, mainly an adequate supply of components.

4.1.2. *The Ministry of Instrument Making, Automation Equipment, and Control Systems (Minpribor)*

In 1974, the Soviet Union and its partners in the RIAD program agreed to start a complementary program to develop minicomputers within CMEA, the so-called SM (*Sistemaia Malaia*, or small system) series. They created a Council for General System Design of Minicomputers, with working subgroups for management information system and computer-assisted design.[45]

The CMEA countries agreed to develop the SM computer family as an extension of Minpribor's existing ASVT (Aggregate System of Computer Technology) computers, such as the M-6000 and M-7000, which were patterned after the PDP-8 and PDP-10 machines. As with RIAD and ASVT, the SM line copied existing Western models, in this case the Hewlett Packard HP-2116 and the Digital Equipment Corporation's PDP-11 minicomputer families.[46]

The SM machines were intended to fill the applications gap where RIAD machines were simply too big or too expensive, especially in process control. The original intent was to start testing initial SM models in 1977.[47] Soviet computer designer Boris Naumov, who had designed the ASVT computers, headed up the SM project.

[45] Rakovskii (1979). The Soviet abbreviation corresponding to ADP and MIS is "ASU," which means "Automated Control Systems," and for CAD it is "SAPR."

[46] Goodman *et al.* (1984).

[47] Naumov (1977).

The SM program initially was a minicomputer program. As the USSR became able to produce more powerful integrated circuits, the "SM" designation appeared on a number of much smaller systems, still preserving HP or DEC compatibility, that the Soviets properly call "microcomputers." In the 1980s, the "SM" designation has also been applied to a new series of Minpribor "personal" computers powered by Minelektronprom clones of Intel 8080 and 8086 microprocessors.

Table VI provides an overview of the SM line of minicomputers. This review does not cover those machines identified as being produced in East European countries, even though they may use Soviet components.

Phase One: 1974–1977; Planning the SM-I. The first phase of SM development laid out the basic designs for the first generation of SM minicomputers ("SM-I") and peripherals, and selected the countries responsible for each aspect of the program.[48] This phase was largely preparatory in nature, in that no SM machine was produced. During this period, however, Minpribor did produce the M400 computer, which operated at 0.1 MOPS, had 64 Kb RAM, and was outfitted with a 5 Mb disk memory unit.

The CMEA member countries reached an agreement to produce four central processors, the SM-1P, SM-2P, SM-3P, and SM-4P, which would form the heart of four models of SM-I computers. The first two were based on Hewlett Packard designs. The latter two were to be upwardly compatible, third-generation 16-bit processors, based on the DEC PDP-11 line of minicomputers.

Agreement was also reached regarding the applications that the SM program would focus on in the development of the minicomputers. These included: control systems for continuous and continuous-discrete technological processes and production; complex scientific experiment control; networks with large minicomputers for data processing in non-industrial applications; and automated design.

In June and July of 1977, Bulgaria, Hungary, GDR, Cuba, Poland, Rumania, USSR, and Czechoslovakia attended the first international tests for SM computers.[49] Unfortunately, no computers were yet ready for inspection. The CMEA commission approved two central processors: the SM-1P (also called SM-2101) and the SM-3P (also called SM-2103). Two internal memory units were approved: the SM-3100 and SM-3101 (latter produced in Poland). In addition, a number of peripheral units received CMEA blessings.

[48] Sources for the information on SM-I include: Ashastin (1980), 81; Kabelevskii (1986), 31; and Lavreniuk *et al.* (1979).

[49] Lavreniuk *et al.* (1979), 122.

TABLE VIA

SELECTED CHARACTERISTICS OF MINPRIBOR SM-I MINICOMPUTERS

Model	SM-1	SM-1M	SM-2	SM-2M	SM-3	SM-4
Generation	SM-I	SM-I+	SM-1	SM-I+	SM-I	SM-1
Year of First Appearance	1978	1981?	1978	1983	1978	1978
Year Serial Production Ended	1987	????	????	????	????	1987
Main Processor						
Soviet model(S) number of CPU	A-131-10 SM-P	A-131-14	A-131-11 SM-2P	A-131-15	SM-3P	SM-4P
CMEA model number(s) of CPU	SM-2101		SM2102	SM-2M	SM-2103	SM-2104
Chip model used in CPU						
Processor cycle time						
Word Length (bits)	16	16	16	16	16	16
Byte size						
Number of addressable registers	4		4	4	8	8
Data types	fixed (8, 16 & 32 bits) floating (32 bits)		fixed (8, 16 & 32 bits) floating (32 bits)	fixed (16 & 32 bits) floating (32 bits)	fixed (16 bits) floating (48 bits)	fixed (8 & 32 bits) floating (32 bits)
Operating speed	400 KOPS		450 KOPS	480 KOPS	220 KOPS	700 KOPS
Selected performance times (μsec)						
Register to register					5	2.1

Fixed point add	2.5	5	2.2	2.1		
Fixed point multiply	36.6	20	10	10		
Floating point add	33	40	18–40	15		
Floating point multiply	110	9	23	52		
Compatibility	HP-3000 M-6000 M-7000, SM-2	HP-3000?	HP-3000 M-6000 M-7000, SM-1	HP-3000? SM-1, SM-2 SM 1210, PS3000	PDP-11 SM-4, SM 1420 SM-1600, SM-1300	PDP-11 SM-3, SM-1420 SM-1300
Price of CPU (thousand rubles)	5		8.7	8.0	3	4.1
Number of processors in system	1	1	1 or 2	1 or 2	1	
Primary Memory						
Maximum capacity (8-bit bytes)	64K	128K	256K	256K	256K	248K
Access time (microseconds)	1200		1200	1200	1200	1200
Length of accessed word (bits)	18					
I/O Channels						
Throughput capacity	250 Kb	4 Mb	250 Kb	2 Mb	800 Kb	800 Kb
Standard Operating Systems	DOSRV		DOSRV, OSRV	ASPO, ROS (multi-processor)	DOS SM, OS SM	FOBOS, OSRV RAFOS
Modes of Operating*	B, RT		B, MP, RT	B, MP, RT, Multiprocessor	B, MP, RT	B, MP, RT
Programming Languages**	Mn, F, A, B		Mn, F, A	F, A, B	MA, F, K, P, B	Mn, F, K, P, B
Price range for a system (000 rubles)	7.7–67.9		27–136.2	20–150	30.7–62.15	46–160

Keys to Abbreviations

* B—Batch; MP—Multiprogramming; VS—Virtual storage; RT—Real time

** AS—ASSEMBLER; F—FORTRAN; PL1—PL/1; C—C language, K—COBOL; RPG—RPG; A—ALGOL, P—PASCAL, Mn—Symbolic code, MA—MACROASSEMBLER

Sources: Artamonov (1988); Kezling (1986); Khatskevich and Protsenko (1988); Ostrovskii (1988); Prokhorov (1987); Prokhorov (1988a,b); Signaevskii (1988); Zonis (1988).

TABLE VIB

SELECTED CHARACTERISTICS OF MINPRIBOR SM-II & SM-III MINICOMPUTERS

Model Alias	SM-1210 SM-53/50	SM-1410	SM-1420	SM-1600	SM-1700
Generation	SM-II	SM-II	SM-II	SM-II	SM-III
Year of First Appearance	1986	198?	1983	1983	1987
Year Serial Production Ended		1983?			
Main Processor					
Soviet model(S) number of CPU					SM-1700 ALP
CMEA model number(s) of CPU		SM-1204	SM-2420	SM-1600.2620	
Chip model used in CPU					K1804VS1
Processor cycle time				360 ns	270 ns
Word Length (bits)	16	16	16	16	32
Byte size	8				8
Number of addressable registers	37		9	8	16
Bytes per operation code					1 or 2
Bytes per instruction	2 or 4			2, 4, 6	1 to 17
Maximum operands per instruction					6
Number of instructions			152	107	304
Number of address modes	4		12		
Data types	fixed (16 & 32 bits) floating (32 & 64 bits) logical		fixed (8, 16 & 32 bits) floating (32 & 64 bits) logical	fixed (8, 16 & 32 bits) floating (32 bits), logical decimal	integer (8–128 bits) floating (32–129 bits) decimal (to 32 digits) character string (to 64Kb) bit field (to 32 bits)
Operating speed	3.3 MOPS	244 KOPS	1–8 MOPS		2.8 MOPS
Selected performance times (μsec)					
Register to register		1	1	2.5	1.2
Fixed point add	0.9		2.8	1.3	
Fixed point multiply	1.8		8.6	9.2	
Floating point add	3.0		11	15	
Floating point multiply	2.5		17	33	

Maximim virtual memory addressable	256 Mbytes				4 gigabytes
Page size					4096 bits
Compatibility	M7000, SM-2M PS3000	PDP-11 SM-4, MIR-2, MIR-3	PDP-11 S, –3, SM-4 SM-1600, SM-1300	PDP-11 SM-3, SM-4 SM-1420, M5000	VAX-780 SM-3, SM-4, SM-1420, SM-1600
Price of CPU (thousand rubles)			4	6	
Number of processors in system	1 or 2			2	
Special & Auxilliary Processors*		SM-2410 MIR		SM-2104.0506 M-5000 compatible	FL
Primary Memory					
Maximum capacity (8-bit bytes)	4 Mbytes	512K	3,940K	1 Mbyte	1–5 Mbytes
Access time (microseconds)	0.54			0.72	0.45
I/O Channels					
Type of data bus (interface)	IUS	OSh	OSh	OSh	OSh
Maximum number of peripherals			20		
Typical hard disk storage capacity (Mb)		4.8		42	242 Mbytes
Standard Operating Systems		RAFOS		DOS SM, OSRV FOBOS DOS SM-1600	MOS VP DEMOS
Modes of Operating**	B, MP, RT, Multiprocessor		B, MP	B, RT	B, MP, RT, VM
Programming Languages***	F, K, P, G MA		Mn, AS, P, B	Mn, F, K, RPG, PL1	F, C, K, P, PL1, B BLISS-32
Price range for a system (000 rubles)		59	63–155		

Keys to Abbreviations

* FL—Floating point; S—Service; M—Matrix; T—Text; MK—Macro Conveyor

** B—Batch; MP—Multiprogramming; VS—Virtual storage; RT—Real time

*** AS—ASSEMBLER; F—FORTRAN; PL1—PL/1; C—C language, K—COBOL; RPG—RPG; A—ALGOL, P—PASCAL, Mn—Symbolic code, MA—MACROASSEMBLER

Sources: Artamonov (1988); Kezling (1986); Khatskevich and Protsenko (1988); Ostrovskii (1988); Prokhorov (1987); Prokhorov (1988a,b); Signaevskii (1988); Zonis (1988).

Phase Two: 1978–1982. Production of SM-I; Introduction of Micros. The actual production of SM-I computers began in this phase.[50] So did the first design and production of SM microcomputers. Soviet pronouncements identified a series of goals for this period, including the increased production of higher-quality and more flexible computers, a wider range of computer applications, improved upward compatibility, and continued development of peripheral components. The SM-I machines were based on printed circuit boards with small or medium scale integration. Kabalevskii (1986) reports that from 1977 to 1980, more than 100 different SM devices were designed, tested, and produced. In addition, 16 operating systems and 14 application packages were produced in the same period.

In 1979, the SM members held the Second International Meeting of the CMEA SM Commission to introduce the following SM-I computers.[51]

SM-1. The SM-1 minicomputer is mainly used in technological process control (ASUTP), both in industrial and laboratory experiments. It can substitute for the earlier M-6000 machine, and is program-compatible with the M-7000, SM-2, and SM-1210. The latter two are discussed below. Operating with up to 64 Kb internal memory, the SM-1 can perform at 0.39 MOPS in register-to-register addition operations. As usual, the speed is quickly reduced during other operations. Fixed-point multiplication is performed at 0.025 MOPS, and division at 0.015 MOPS. The SM-1 reportedly can be equipped with an array of peripheral equipment, including external disk memory from 860 Kb to 5 Mb. One available tape drive can store up to 100 Mb of data.

The SM-1 computer's operating system is DOS RV, which supports MNEMOCODE, a symbolic programming language, as well as FORTRAN, ALGOL, MACRO, and BASIC.

SM-2. Used in automatic process control systems, equipment testing, communication links, and engineering calculations, the SM-2 is fully compatible with the earlier M-7000, and program compatible with the M-6000, SM-1, SM-2M, SM-1210 and PS-3000. Utilizing the SM-2P processor, which follows the HP line discussed above, the SM-2 can perform 0.45 MOPS of fixed-point addition, 0.1 MOPS fixed-point multiplication, between 0.025 and 0.055 MOPS of floating-point addition, and 0.043 MOPS of floating-point multiplication. The computers internal memory can range between 64 Kb and 256 Kb.

[50] For more information on the production of SM-I computers see Artamonov (1988) and Kezling (1986).

[51] Riabov (1981).

The SM-2 employs the DOS RV and OS RV operating systems, which can handle a symbolic program language, MNEMOCODE, as well as FORTRAN and ALGOL.

SM-1M. Sometimes referred to as the "second generation" of the SM-1, the SM-1M employs a slightly improved processor and has a larger operating memory (128 Kb) and program memory than its predecessor. It is fully compatible with the SM-1 and SM-2 computers, and program compatible with the M-6000 and M-7000 as well. The SM-1M can perform 0.2 MOPS of fixed-point addition, 0.05 fixed-point multiplication, 0.025 floating-point addition, and 0.11 MOPS floating-point multiplication. Some configurations of this machine utilize the SM-5211 cassette tape device for external memory.

SM-2M. Just as the SM-1M is an improvement of the SM-1, the SM-2M is a "second generation" of the SM-2. Continuing along the HP path, the SM-2M uses two central processors in applications such as process control in energy and metallurgy, as well as ADP in small firms. With an internal memory that can range between 64 Kb and 256 Kb, the SM-2M can perform fixed-point addition at 0.48 MOPS, fixed-point multiplication at 0.1 MOPS, floating-point addition between 0.025 and 0.055 MOPS, and floating-point multiplication at 0.043 MOPS.

The SM-2M uses either the ASPO or ROS operating systems, and the program languages FORTRAN, ALGOL, and BASIC.

SM-3. The first of Minpribor's DEC PDP-11 line of computers to appear in the late 1970s was the SM-3. Intended for monitoring scientific experiments, equipment testing and controlling, and various calculation duties, the SM-3 was also employed in multi-machine systems with RIAD computers, serving as a remote terminal, peripheral processor, input-output processor, and in other network capacities. The computer was shipped in eight standard configurations, depending upon final application.

Operating at an average 0.126 MOPS, the SM-3 could also attain 0.2 MOPS in the faster register-to-register operations. The machine's internal memory could range from 16 Kb to 56 Kb, and was program compatible with the SM-4, SM-1420, SM-1600, and SM-1300. It operated under the DOS SM and OS SM operating systems, and could use the program languages MACROASSEMBLER, FORTRAN, COBOL, PASCAL, and BASIC. A later version of this computer, the SM-3-20, was serially produced beginning in 1980.

A distinguishing feature of the SM-3 and SM-4 (i.e., the DEC-like) families of minicomputers is their use of a standardized data bus, the *Obshchaia Shina* ("OSh") which is the functional equivalent of DEC's UNIBUS. Like

UNIBUS, the OSh is a parallel bus consisting of 56 data lines and is designed to promote compatibility across a wide range of computers and peripheral devices.

SM-4. Minpribor's SM-4 has developed into the workhorse of Soviet industrial, research, and design applications, and boasts the largest installed base of all Soviet minicomputers. Intended for use as an automated design workstation, technological process controller, or scientific experiment manager, it was first announced in 1978, and entered production at Minpribor's Kiev Elektronmash complex in 1980. Naumov once reported that the SM-4 was four times faster than the SM-3, and twice as expensive. The available scattered evidence generally supports Naumov's assertion.

The SM-4 computers have at least 64 Kb and no more than 256 Kb of internal memory. They operate at an average 0.244 MOPS (Gibson-1 test), but can reach 0.7 MOPS in register-to-register operations, though there is some discrepancy in official Soviet statistics. The SM-4 computers are program compatible with the SM-3, SM-1420, and SM-1300 computers. They run the FOBOS, OS RV, and RAFOS operating systems. They can accommodate the following program languages: MNEMOCODE, FORTRAN, COBOL, PASCAL, and BASIC.

Minpribor produces the SM-4 computer in at least seven different configuration subfamilies each of which bears its own identification number ranging from SM-1401 to SM-1407. To add even more confusion, the SM-1403 is also referred to as the SM 52/11, and the SM-1404 is alternately known as the SM 51/13. The general characteristics and distinctions of these subfamilies follow immediately below.

The SM-1401 has at least eight common configurations, which differ mainly by the size of internal and external memory. The SM-1401s operate under the FOBOS operating system.

The SM-1402 comes in at least two configurations, with the only distinction being the number of internal units used to achieve 64 Kb of memory. These use the DOS operating system.

The SM-1403 (also known as the SM-52/11) is shipped in at least eight different configurations that differ by the type of external memory and printer devices installed with the computer. These machines operate under OS RV.

The SM-1404 (also known as the SM-51/13) is a version of the SM-4 computer that comes in two different configurations, and each has two processors, which sets the SM-1404 apart from other members of the SM-4 family. It also has two transistor internal memory devices and can access up to 29 Mb on attached fixed disk units. Like the SM-1403, it operates under OS RV.

The SM-1405 computer has been identified in at least five different configurations which vary somewhat by the type and size of internal memory, but mostly by the communications devices employed. Noted applications include automated systems of scientific experiments and process control systems, and it utilizes both the FOBOS and OS RV operating systems.

The SM-1406 is configured two ways and used mainly for database processing. It operates under OS RV and DIAMS.

The SM-1407 has been encountered in three configurations, both intended for automated workstation applications and employing the OS RV operating system.

The year 1979 marked the appearance of the first SM microcomputer systems, the SM-50 family.[52] The model numbering system for this set of SM machines was more elaborate and sometimes more confusing than for the first. In general, most machines of this period bear a 50 series number. The SM-50 class computer is a microcomputer system used for numeric control, scientific measuring, and network terminals. The SM-51 class marks a further development of the SM-1 and aimed for compatibility with the earlier machines. The SM-52 includes multiprocessor systems that can be used in conjunction with RIAD computers. Multiprocessor and multimachine computers fall under the SM-53 line, and the SM-54 involves specialized processors for matrix operations, speech synthesis, seismic studies, etc. In addition, a host of peripheral devices were designed and produced.

Phase Three: 1983–1986. Production of SM-II; Plans for SM-III. In 1983, Minpribor began production of the SM-1420 and SM-1600 minicomputers. These represent a continuation of the 16-bit, PDP-11-compatible, SM-4 computer line and the beginning of SM-II.

The ministry's general pronouncements regarding the progress and prospects of the SM program at this time continue the usual exhortations for improved quality. Hard disk memory devices were singled out for particular emphasis, as was the need for improvements in servicing the installed computer base.[53] Soviet analysts and scientists emphasized that greater speeds and memory storage were needed for improved CAD and scientific research applications. LSI circuits were to be used.[54] In the late 1970s, the Digital Equipment Corporation had begun to ship its very successful 32-bit VAX/780 computers. It was not accidental, therefore, that this period marked the

[52] Naumov (1980).

[53] Zavartseva and Ivanova (1986), 44.

[54] Prokhorov (1987), 8.

beginning of Minpribor's attempts to develop VAX-like 32-bit machines compatible with the SM-4 minicomputer.[55]

Minpribor's SM-II computers included the following:

SM-1210 (SM-53/50). Appearing in the mid 1980s as a continuation of the HP-like SM line, the SM-1210 employs two central processors and can operate in either dual-processor mode or dual-machine mode, meaning that the processors can work in tandem with the same memory, or divide the memory between them and operate separately. The second processor is the input/output processor from the SM-50/60 computer. The SM-1210 is used in management information systems and process control applications. The speed of the machine is difficult to assess, since published Soviet sources give different numbers. For example, fixed-point addition is rated at either 3.3 MOPS or 1.1 MOPS, a substantial discrepancy that is repeated for various operations. In any case, this machine appears to rate above the earlier SM-1 and SM-2 models. The SM-1210 can utilize between 2 Mb and 4 Mb of internal memory.

The SM-1210 is program compatible with the M-7000, SM-2M, and PS-3000. It uses the OS SM 1210 operating system, and supports FORTRAN, COBOL, PASCAL, BASIC, and MACROASSEMBLER.

SM-1410. Used in automated systems of scientific experiments and for numeric program control, the SM-1410 comes in three configurations, with the third employing a special language processor. Operating at an average 0.24 MOPS but capable of 1 MOPS in register to register operations, the SM-1410 can be installed with 64 to 512 Kb of memory.

SM-1420 (SM 51/20). In 1983, the Kiev Elektronmash plant began serial production of the SM-1420 which marks a further development of the SM-4 line of computers. Appearing in at least 12 different configurations designed for different applications, this computer is used in data processing, scientific experiment control, scientific and economic calculations, and networks. It is both input-output compatible and program compatible with the SM-3 and SM-4 computers, and program compatible with the SM-1600 and SM-1300.

The machine can access between 248 Kb and nearly 2 Mb of internal memory, and usually has 4 Mb of external memory. It is clocked at 1 MOPS during register-to-register fixed-point addition, but with a "special algorithm" it reportedly can achieve 8 MOPS. The SM-1420 runs under three different operating systems: OS RV 2.0, RAFOS, and ROS RV. The following pro-

[55] Zavartseva and Ivanova (1986).

gram languages are available: MNEMOCODE, ASSEMBLER, PASCAL, and BASIC.

SM-1600. The SM-1600 is used for statistical and planning purposes, trade management applications, as well as banking, transport, agriculture and small industrial enterprises. In a rather strange design that apparently seeks to take advantage of software written for the earlier generation M-5000 computer, this machine employs two processors. The first is the same one used in the SM-1420, ensuring SM-4 compatibility. The second processor actually comes from the M-5000 computer line which is compatible with the PDP-11's forerunner, the PDP-8.

Operating at 0.045 MOPS during addition operations, the SM-1600 can utilize between 256 Kb and 1 Mb of operating memory. Typical installations employ three disk drives of 14 Mb each, and one tape drive with a 10.24 Mb capacity.

Phase Four: 1987 and After. Production of SM-III. The VAX-compatible SM-1700 marks the advent of SM-III and the development of more powerful Minpribor minicomputers that break the barrier of memory limitations through virtual memory machines that can address up to four gigabytes of data. Often referred to in the Soviet literature as the first model of the "highly productive" 32-bit Soviet machines, the SM-1700 entered production in September 1987 at the Sigma Production Association in Vilnius. The announced plans were to produce "several scores."[56] This machine is intended for CAD, flexible manufacturing systems, planning calculations, scientific research applications, and automatic process control systems. According to one source, the SM-1700 can perform 2.8 MOPS during "short operations," but only 0.3 MOPS using the Whetstone benchmarks.[57]

The Soviets managed to retain compatibility with earlier machines by giving the SM-1700 the ability to emulate the earlier 16-bit processors' command set. Thus, the new VAX-compatible SM-1700 can still run the vast library of software created for the PDP-11, SM-3, SM-4, SM-1420 and SM-1600 computers. In addition, the SM-1700 retains the same hardware interface as the earlier models, so it can use the SM family peripherals and read the same data files.

The SM-1700 is typically outfitted with two disk drives (SM-5504) that can handle 121 Mb each, a tape drive unit that stores 40 Mb, as well as a host of smaller external memory devices.

[56] Moscow Radio, September 23, 1987, as reported in FBIS-SOV-87-186, 54, September 9, 1987.

[57] Prokhorov (1988a), 7.

TABLE VII

MATCHING MINPRIBOR SM COMPUTERS WITH HP AND DEC COUNTERPARTS

Model	Country	Year shipped	Closest U.S. counterpart	Year U.S. shipped	SM lag in years
SM-I					
SM-3	USSR	1978	DECPDP11/20	1970	8
SM-4	USSR	1978	DECPDP11/20	1970	8
SM-II					
SM-1410	USSR	1983?	DECPDP11/45	1972	8+
SM-1420	USSR	1983	DECPDP11/45	1972	11
SM-III					
SM-1700	USSR	1987	DECVAX11/780	1978	9

Sources: Artamanov (1988); Kezling (1986); Phister (1979).

Comparing SM Systems With DEC and HP Originals. One way of evaluating Soviet success or failure in computing is to compare Soviet with Western achievements. While this assuredly is not the only standard that could be applied, or even the best one, it has the advantage of being meaningful to Western readers. It has the additional advantage of being feasible inasmuch as it relies neither on dubious Soviet measures of effectiveness nor on *ad hoc* reportage of scattered cases. Even so, direct comparisons are not always easy to make because Minpribor computer designers have less slavishly followed Western designs than their mainframe colleagues at Minradioprom. At the risk, therefore, that some apples may be set alongside some oranges, the following comparisons are offered. (Also, see Table VII.)

In the SM-I "generation" of machines, the SM-3 is roughly equivalent to the DEC PDP 11/20. Both operated at about 0.2 MOPS, and both employed about the same size operating memory. The DEC machine first entered production in 1970, and the Soviet computer in 1978.

The SM-II "generation" involves a slightly more complicated comparison, but the measured result is about the same. Both the SM-1410 and the SM-1420 are compared with the PDP 11/45. All of the machines operate at approximately the same speed and are used for similar tasks, but the SM-1420 has larger internal memory capabilities. The SM-1410 apparently was introduced in 1983 along with the SM-1420, which would place it approximately 11 years behind its DEC counterpart.

The SM-III generation marked the advent of VAX-compatible Soviet computers. The SM-1700 is roughly comparable with the VAX 11/780, which appeared in the late 1970s. The SM-1700's appearance in 1987 places it approximately eight years behind DEC.

This comparison of Minpribor SM minicomputers with their DEC counterparts in each "generation" of the SM line finds that Minpribor lagged about eight years in the late 1970s, and today lags by about the same interval or slightly more. Unfortunately, it has not been possible to make comparisons of performance, but all evidence suggests that SM computers substantially underperform the "comparable" DEC and HP machines.

The differences between the SM machines and their American counterparts are particularly notable at the component level. Whereas 32-bit single chip processors are commonplace at DEC and HP, Minpribor continues to rely on Minelektronprom's bit-slice chips such as the 4-bit K1804VS1.[58]

Minpribor Microcomputers. In the 1980s, Minpribor has brought out a variety of microcomputers under both the SM and the ISKRA labels. Generally, the SM machines are intended for technical and scientific applications while the ISKRA machines are more likely to be professional workstations for planners, accountants, etc. Basic data on most of them are provided in Table VIII and Table IX. Additional remarks about the most important machines are described below.

The SM-1300 is a microcomputer version of Minpribor's SM-4 (DEC) line and said to be roughly equivalent to the PDP 11/03 or PDP 11/04. It uses the 8-bit KR1802 bit-sliced ALU and is software compatible with the SM-4 and SM-1420. The basic box weighs less than 16 lbs and can be configured with peripherals in multiple ways including as a CAD workstation and LAN server.[59]

The SM-1800 family, which appeared in 1981, was one of the first Minpribor microcomputers. Its purpose was to replace Minpribor's old SM-I and even older M40 and M60 machines in laboratory, process control, and data preparation applications. Based on Minelektronprom's KR580 copy of Intel's 8080 8-bit microprocessor, the SM-1800 comprises more than 40 modules. Submodels abound. For example, the SM-1801 is a bare-bones processor box, the SM-1802 is equipped with laboratory control devices, the SM-1803 is for industrial process control and is shipped in at least nine major configurations, the SM-1804 is designed for operation in adverse environments, etc. Prices range from 20 to 34.6 thousand rubles ($33K to $58K).[60]

[58] Ostrovskii (1988).

[59] Information on the SM-1300 is from Kezling (1986), 513–517; *Elorg* (1986), 6; Kuznetsov *et al.* (1988), 78.

[60] Information on the SM-1800 is from Prokhorov and Smirnov (1986), 9; Shkamarda (1986), 6; Prokhorov and Landau (1984), 28; Giglavyi *et al.* (1984), 33; Kezling (1986), 541–558; Kuleshova (1987), 88; Rukavishnikov (1988); Oprishko *et al.* (1987), 40; Ivanova (1987), 40; Grevtsev (1988), 41.

TABLE VIII

MINPRIBOR SM-LINE MICROCOMPUTERS

Minpribor name also known as	SM-1300	SM-1625	SM 50/60 SM-1634	SM-1800	SM-1810
Year First Produced				1981	1986
Chip	KR1802	KR580?	K589	KR580IK80A	KM1810VM86
foreign analog		Intel 8080	Intel 3000	Intel 8080	Intel 8086
Speed (KOPS)					
reg-reg	400–500	100–500	20–170	125–500	2500
Word length	16	8	16	8	16
RAM (Kbytes)	64–256	64	16–128	64	256
ROM (Kbytes)		4	8–16	2–10	
Max. Addressable Space (Kbytes)	256	64		64	4 Mbytes
Number of Commands	SM-3, SM-4 set			78	135
Operating Systems	OS RV, RAFOS, DIAMS		DOS ASPO	DOS 1800, MOS RV, OS 1800	DOS-16, ADOS, DOS 1810
Program Languages	Usual SM set.		ALGOL, F-II, F-IV, B, As	F, PL/M, B C (subset) Mi (subset)	All MS-DOS languages
Compatibility	SM-4 family	SM	SM-1, SM-2 PS-X000, SM-1210	CP/M software	IBM-PC
Type of Data Bus	OSh	I41		I41	I41
Applications	ARM, SAP LAN server	Process, lab, & network control	Terminal, Process control	Professional Workstation	Professional Workstation

Keys to Abbreviations

Program Languages: As—ASSEMBLER; B—BASIC; C—C; F—FORTRAN; F-II—FORTRAN II; F-IV—FORTRAN IV; I—IAMB; Ma—MACROASSEMBLER; Mi—MIBOL; P—PASCAL

Sources: Abramovich *et al.* (1985); Artamanov (1988); Elorg (1986); Giglavyi *et al.* (1984); Iaroshevskaia (1986); Kezling (1986); Prokhorov and Landau (1984); Prokhorov and Smirnov (1986); Savel'ev (1987a); Shkamarda (1986).

TABLE IX

MINPRIBOR ISKRA-SERIES MICROCOMPUTERS

Minpribor name	ISKRA 226	ISKRA 555	ISKRA 2106	ISKRA 1030
Year First Produced	1981			1987
Chip	Dual KR580IK80A	KR589IK02	KR580IK80A	KM1810VM86
foreign analog	Intel 8080	Intel 3000	Intel 8080	Intel 8086
Speed (KOPS)				
reg-reg	600	650	250–400	1000
Word length	16	16	16	16
RAM (Kbytes)	128	16–48	4–16	256
ROM (Kbytes)	16–24	20–28	8–16	
Number of Commands	95			
Operating Systems	OS ISKRA, DOS	OS ISKRA	OS ISKRA	ADOS
Program Languages	As, B, F	I	I	B, Ma, P, C, I
Compatibility	Wang 2200	Iskra SM (data)	Iskra	MS DOS
Applications	Ec. planning ARM, Science lab control, ASU, ASU TP	Ec. planning ARM, ASU	Ec. planning ARM, ASU	Professional Workstation

Keys to Abbreviations

Program Languages: As—ASSEMBLER; B—BASIC; C—C; F—FORTRAN; F-II—FORTRAN II; F-IV—FORTRAN IV; I—IAMB; Ma—MACROASSEMBLER; Mi—MIBOL; P—PASCAL

Sources: Abramovich *et al.* (1985); Artamanov (1988); Elorg (1986); Giglavyi *et al.* (1984); Iaroshevskaia (1986); Kezling (1986); Prokhorov and Landau (1984); Prokhorov and Smirnov (1986); Savel'ev (1987a); Shkamarda (1986).

The SM-1800's main operating system is DOS-1800, an adaption of CP/M. In addition to familiar programming languages running under CP/M, a Russian version of WORDSTAR named TEXT is available.

Old-fashioned Minpribor printed circuit board fabrication has made for substandard quality product. Nevertheless, Minpribor has produced fairly large numbers of the SM-1800 and various versions of the machine have found relatively widespread application in Soviet industry. Users complain about chronic shortages of peripherals necessary for proper configurations.

The SM-1810 is an updated, 16-bit, version of the SM-1800. It uses Minelektronprom's KM1810 processsor which is a clone of the Intel 8086 chip. It is said to be an order of magnitude faster than its predecessor. Production began in 1986. An "industrial version" is produced as the SM-1814.[61]

The ISKRA 1030 may be a second Minpribor IBM-PC compatible or it may be another brand name for the SM-1800. In any case, it is made by the Kursk Schetmash factory and is targeted at the traditional users of ISKRA computers and bookkeeping machines, particularly planning agencies and central ministries. Like the SM-1800, this machine also uses the K1810VM86 microprocessor. Ministry sources say that it is to be shipped in three basic configurations. The ISKRA 1030.11, which is the base model, is a dual floppy, 256 Kb machine. Another model, the ISKRA 1031 is presumably equipped with a hard disk. The operating system is ADOS which is said to be MS-DOS compatible.[62]

The ISKRA 226 is a WANG-2200 work-alike. The first half dozen WANG-2200 machines produced were exported to the Soviet Union in 1972 and 1973 and many more followed them during the remainder of the decade. In total, about 2000 were shipped to the USSR and Eastern Europe and they became very popular as planners' workstations. The tightening of U.S. and COCOM export restrictions in the wake of the Afghanistan invasion not only ended WANG's exports but deprived Gosplan and other central planning agencies of their supplier.[63]

Minpribor's "Schetmash" factory in Kursk had been producing a variety of ISKRA bookkeeping machines in the 1970s. On the basis of that experience,

[61] Information on the SM-1810 is from Prokhorov and Smirnov (1986), 9; Kuleshova (1987), 88; *Elorg* (1986), 13; Korneichuk *et al.* (1986), 11.

[62] Iaroshevskaia (1986), 23.

[63] Information on the ISKRA 226 is from Artamonov (1988), 207; Nikitin and Ostrovskii (1988); Abramovich *et al.* (1985), 35; Poom *et al.* (1986); Sasov (1986), 53; Krilov *et al.* (1985), 43; and the senior author's observations.

Minpribor designers reverse engineered the WANG-2200 as the ISKRA 226, which entered serial production in 1981. The first copies of the new machine went to the Academy of Sciences and Gosplan and by 1985 more than 800 ministries and departments were using it. The ISKRA 226.7 is shipped with a variety of Soviet, Bulgarian, and East German peripherals in at least seven main configurations. Their prices vary between 11 and 25 thousand rubles ($18000–$42000). Most of the software has been developed by Gosplan. So pleased were the authorities with Minpribor's work that Academician Velikhov nominated the machine's designers for the 1985 State Prize.

4.1.3. *The Ministry of the Electronics Industry (Minelektronprom)*

Minelektronprom is the monopoly producer of electronic components in the Soviet Union. It is a "VPK" ministry, i.e., it is an officially designated member of the "military-industrial complex." A long-time producer of military computing systems, Minelektronprom has become a major supplier of civilian computers in the 1980s. It produces a huge array of computing devices, most of them bearing the "ELEKTRONIKA" brand name, and only the most important are surveyed here.[64]

ELEKTRONIKA Minicomputers. Minelektronprom offers a family of PDP-11 compatible, 16-bit minicomputer systems that are hardware and software compatible. They include: the ELEKTRONIKA 100-16 at the bottom end, the ELEKTRONIKA 100-25 in the middle range, and the ELEKTRONIKA 79 at the top end. Basic information on each is given in Table X.

These ELEKTRONIKA minicomputers are software compatible with Minpribor's SM-3 and SM-4 families, and may be configured with SM peripherals. They also share the OSh data bus as well as SM systems and applications software.[65]

The ELEKTRONIKA 100-16 can accommodate a limited range of peripherals. Its memory ranges from 8 to 56 Kb in blocks of 16 Kb ferrite core memory. The ELEKTRONIKA 100-25 offers greater storage—up to 248 Kb, which Soviet sources claim give it the ability to work in multiprogram mode. A wider selection of peripherals is available and its multiplexor can handle up

[64] The two best sources that we have encountered on Minelektronprom computers are Glushkova and Ivanov (1986) and Tolstykh *et al.* (1987).

[65] For information on the ELEKTRONIKA minicomputers, see Savel'ev (1987b), 114; Iakubaitis (1985), 183; Goodman (1984); Kezling (1986), 629–635; and Verner *et al.* (1986), 7.

TABLE X

BASIC OPERATING CHARACTERISTICS OF THE ELEKTRONIKA MINICOMPUTERS

Characteristic	Model 100-16	Model 100-25	Model 79
Word Size	16	16	16
Max Main Memory (Kbytes)	56	248	4088 (2)
Data types	fixed	fixed/float	fixed/float
Precision, float, bits	32	32 & 64	
Number of Commands	73	89	137
Speed/MOPS	0.25	0.8	3
General Purpose Registers	8	8	16
Process Work Regime	User	User & Supervisor	User, Supervisor, & Internal
Price (rubles)			12,000

Source: Tolstykh *et al.* (1987); Veitsman (1988).

to 16 terminals. Recent Soviet sources of the late 1980s state that it is being used in CAD and R&D applications. The ELEKTRONIKA 79 command set includes 56 additional commands of a floating-point processor and it has a high-speed buffer of 2 Kb capacity. It supports up to eight external memory devices.

No production data are available on these ELEKTRONIKA minicomputers, but frequent references to them in the Soviet literature indicate that they are relatively abundant. The indications are that the ELEKTRONIKA minis originated in the 1970s as a line of military computers that are now seeing such civilian applications as data processing, data base management, automatic control systems, and scientific research. They are produced at the Electronic Computer and Control Machines Plant (VUM) in Kiev.

ELEKTRONIKA Microcomputers. Minelektronprom has produced a wide assortment of military and civilian microcomputers. Table XI provides technical details on many of these, and the most important are discussed below.

The ELEKTRONIKA 60 family. The ELEKTRONIKA 60 and its descendants are probably the most numerous family of computers in the Soviet Union. These are 16-bit machines, mutually compatible and software compatible with the ELEKTRONIKA 100/25 and with Minpribor's SM-3, SM-4

TABLE XI

MINELEKTRONPROM MICROCOMPUTERS

Elektronika name also known as	60	60M	60-1 MS-1211.01	60-1 MS-1211.02	60-1 MS-1212	80-1 MS-1213	85 MS 0585
Year Produced							1987 (?)
Chip	K581	K581	K1811	K1811	K1811	K1804	K1811
aka	M1	M2	M6 MS 1601	M6 MS 1601	M6 MS 1601	M5	
Speed (KOPS)							
reg-reg		250	500	500	600	800	600
Word length	16	16	16	16	16	16	16
RAM (Kbytes)	8	8		128	256	248	512
ROM (Kbytes)		4	48	48	48	48	
Max. Addressable Space (Kbytes)	64	64	256	256–1000	4 Mbytes	256–4096	4 Mbytes
Number of Commands		81	138	138	138	95	138
Operating Systems	RAFOS	PLOS, FODOS, TMOS	PLOS, FODOS, TMOS	PLOS, FODOS, TMOS	PLOS, FODOS, TMOS	FODOS, MDOS TMOS	PROS, FODOS, DEMOS, MOS-80, MIKRO-80, SP-80
Program Languages		AS, F, B		MA, F		AS, F, B	F, P, B, K, MA, M2 MA, M2
Compatibility	SM	SM	SM	SM	SM	SM	SM

(*continues*)

TABLE XI (*Continued*)

Elektronika name also known as	K1-20 MS-2702	S5-21M	S5-41 MS12102.1	BK-0010	DVK-2M NTs-80-20/x	DVK-3M2 NTs-80-20/x	DVK-4 NTs-80-20/x	T3-29M
Year Produced					1981–4	1984–	1986–	1983/4
Chip	KR580	K586	K1801VM1	K1801VM1	K1801VM1	K1801VM2	K1801VM3	K589
aka				MS 1201.01	MS 1201.01	MS 1201.02		
Speed (KOPS)								
reg-reg	500	200	500	500	500	1000	1200	500
Word length	8	16	16	16	16	16	16	16
RAM (Kbytes)	1	0.5	2	32	56	64–248	64–248	128–25
ROM (Kbytes)	8	6	16	32	8	8	?	64
Max. Addressable Space (Kbytes)	64	64	64	64	64	64–4096	64–4096	2048
Number of Commands	78	256	64	64	64	64 or 72	64 or 72	139
Operating Systems	Resident	DS-81 DOS	Resident		OSDVK	OSDVK	OSDV	Resident
Program Languages	As			Fk, B	M, B, F, P, M2	AS, B, F, P	M, B, F, P, M2	AS, B
Compatibility	Not SM	Not SM	SM	SM	SM	SM	SM	HP 98xx

Keys to Abbreviations:
Operating Systems: PLOS—Punched-Tape Op Sys; FODOS—Background Disk Op Sys; TMOS—Test Monitoring Op Sys; DEMOS—OS UNIX 2.9 Bell Labs
Programming Languages: As—ASSEMBLER; F—FORTRAN; B—BASIC; K—COBOL; M—MACRO; P—PASCAL; M2—MODULA-2; Fk—FOKAL
Sources: Elorg (1986); Glushkova and Ivanov (1986); Khatskevich and Protsenko (1988); Kokorin *et al.* (1986); Lopatin *et al.* (1985); Murenko *et al.* (1986); Popov *et al.* (1984); Tolstykh *et al.* (1987).

families of minicomputers. It represents, obviously, yet another Soviet derivative of DEC's PDP-11.[66]

The ELEKTRONIKA 60 appeared first around 1980, probably as the outgrowth of an effort to supply the Soviet military with small-scale computers. This machine has found widespread employment in control devices of all sorts, computer-controlled machine tools, etc. Despite its limited capacity (250 KOPS, 8 Kb RAM, 4 Kb ROM), the ELEKTRONIKA 60 has been fitted with peripherals and asked, in the 1980s, to play roles ranging from that of network terminal to CAD workstation. Modestly enhanced versions, such as the multi-board ELEKTRONIKA 60M, ELEKTRONIKA 60-1, ELEKTRONIKA 85, and MS-121X machines are currently produced and widely used in Soviet science and industry. Single-board members of the family include the ELEKTRONIKA-41, ELEKTRONIKA NTs 80-01D (alias MS1201.OX). These machines use the NMOS K581, K1801, and K1811 chips.

An important group of ELEKTRONIKA 60 descendants is the modular DVK (*Dialogovii Vychislitel'nyi Kompleks*, which translates as Interactive Computer Complex). The DVK-1, DVK-2 (1983) and DVK-2M (1984) are table-top systems intended primarily for classroom use. Both consist of an ELEKTRONIKA NTs 80-01D computer using the K1801VM1 processor configured with 56 Kb RAM and character monitor. The DVK-1 consists of nothing else and is intended as a student workstation. The DVK-2 and DVK-2M are also equipped with a 512 Kb floppy disk and printer to serve as teacher workstations in classroom networks. The DVK-4 (1985) and DVK (1986) are equipped with from 64 Kb to 4 Mb of RAM, 440 Kb or 800 Kb floppy disk drives, monochromatic graphics monitor, and graphics plotter. The DVK (1987) offers a color graphics monitor.

Minelektronprom claims considerable versatility for its DVK machines. For example, K580VM80A and K1801VM86 processor boards are said to be available for the DVK machines which would, theoretically, provide software compatibility with the ES 1840, ISKRA 1030.11 and other CP/M or CP/M-86 machines. No reports of using such coprocessors have been encountered in the literature. A UNIX operating system was reported under development in 1986. Reliability is a major problem for the DVK systems. Although the mean time before failure is claimed to be 3000 hours for all DVK systems, reports from the field indicate that it is far less. One Karangada computer instructor, responsible for teaching high school teachers the elements of computing,

[66] Information on the ELEKTRONIKA 60 family comes from Glushkova and Ivanov (1986); Kokorin *et al.* (1986); Tolstykh *et al.* (1987); Kezling (1986); Grigor'ev (1987), 23; Legavko and Vasilenko (1988); Savinov and Gritsyk (1988), 33; and Lopatin *et al.* (1985).

recently complained bitterly that her institution's DVK-2M crashed daily and caused her much embarrassment before the teacher trainees.[67]

Another member of this family is the ELEKTRONIKA BK-0010 which Minelektronprom bills as the Soviet Union's "first home computer." This little machine is essentially identical to the DVK-1 described above and fulfills the same role in classroom computer networks. Output began in 1985 at Minelektronprom's "Eksiton" factory in Pavlovskii Posak. Some 20,000 reportedly were produced in 1987 and an equal number were to be produced in 1988. Priced at 650 rubles (about $1000 at the official exchange rate or $135 at the black market rate), some 194 units were sold at the Ministry's ELEKTRONIKA retail store in Moscow in 1985. In 1986, sales were planned to be 2000 units.[68]

A visit to the ELEKTRONIKA store in the summer of 1988 found the BK-0010 on display, but would-be customers were barred from touching it. No sales personnel were available and no machines were available for purchase. Customers were invited to enter their names on a waiting list but no indication was given of how long the waiting period might be. The same ELEKTRONIKA store was quoting a four-year wait for videocassette recorders.

The vast majority of BK-0010 machines are going to classroom computer laboratories. From 10 to 15 BK-0010s serve as student workstations networked to a teacher's DVK-2M. This configuration, termed the "KUVT-86," was Minelektronprom's 1987 response to the Soviet school system's cry for educational computers. But it is so inadequate and performs so poorly in practice that it bodes to give computing a bad reputation with Soviet high school students.[69]

The most recent version of this educational system, the one intended to compete with Minradioprom's KORVET, is called the "UKNTs" and made at the Elektronmash factory in Moscow.[70] This system differs in certain minor respects from the earlier versions. It features the dual processor DVK-3 (alias ELEKTRONIKA NTs80-01D) at the teacher's desk. The first, dubbed the "central machine," has 64 Kb of RAM and a K1801VM2 processor said to be capable of 800 register-to-register operations per second and operates as server to a 57,600 baud ring network connecting student workstations. The teacher's second machine, called the "peripheral machine," is like the first except with only 32 Kb RAM and is intended to control peripherals such as the keyboard, bit-mapped monitor, tape cassette recorder, sound generator,

[67] Koisina (1988).
[68] Shekhovtsev (1988), 126; and Gorelov (1988).
[69] See, for example, Denisenko (1986) and Koisina (1988).
[70] Information on the ELEKTRONIKA UKNTs is from Driga (1986) and Polosin *et al.* (1986).

printer, and 400 Kb or 800 Kb floppy disk drives. Student workstations look very similar to the BK-0010.

Other ELEKTRONIKA microcomputers. Minelektronprom manufactures a variety of other small computers intended mainly as imbedded automatic controllers. Examples of these are the ELEKTRONIKA S5 machines. The S5-21 controller uses the K586 chip, which has no known foreign analog and is incompatible with other known Soviet computers. It is a 16-bit machine with small (0.5 Kb RAM and 6 Kb ROM) memory. The slightly larger S5-41 uses the ELEKTRONIKA 60 instruction set.

The ministry also makes the ELEKTRONIKA K series of computers. These are small KR580-based systems (8080 work-alikes) with very small memory which are used in equipment controllers, testing devices, checkout equipment, etc. The most recent of these that we have encountered is the ELEKTRONIKA K1-20 (alias MX2702).

The story of the ELEKTRONIKA-60 and its descendants strikingly illustrates a fundamental characteristic of Minelektronprom's approach to computer technology. That approach is one of tiny, incremental changes in existing products accompanied by name changes and price increases. Whether we look at the basic components produced by Minelektronprom or its computer systems, an identical picture of extreme technological conservatism emerges. It is remarkable that the ministry has been able to preserve such a fundamentally unresponsive posture in the face of users' entreaties and complaints, of prestigious commissions' proddings, and even of the Politburo's decrees.

"Home Brew" and Other Soviet Microcomputers. A variety of other microcomputers have appeared in the 1980s. Most of these are based on Minelektronprom's KR580 or KM1810 copies of Intel's 8080 and 8086 microprocessors. The development of these machines indicates that many organizations and individuals in the USSR have the technical ability to "home brew" computers using off-the-shelf components. On the other hand, it also indicates that many users and would-be users find their needs unmet by the computers supplied through normal channels and feel it necessary to develop their own equipment. Soviet research laboratories, in particular, seem to have been forced to develop micros for themselves. Alternatively, some "home brewers" may have worked for the fun of it. Table XII gives an overview of some of the machines identified in Soviet computer periodicals. There are some interesting stories behind many of these machines.

NEIRON 19.66. This IBM PC/XT compatible is manufactured by the Ministry of the Communications Equipment Industry (Minpromsviazi) and

TABLE XII

OTHER MICROCOMPUTERS

Name	IRISHA	OKEAN 240	NEIRON I9.66	MIKROSHA	KVANT	KRISTA
Year First Produced				1986		1987
Chip	KR580VM86	K580VM80	K1810VM86	KR580VM80A	KR581	K580VM80A
foreign analog	Intel 8080		Intel 8086	Intel 8080		
Speed (KOPS)						
reg-reg		600	1000			
Word length	8	8	16	8	16	8
RAM (Kbytes)	48–128	128	256–1 Mbyte	32+	256–4 Mbyte	32+
ROM (Kbytes)	4–64	16		2+		2+
Max. Addressable Space (Kbytes)			1 Mbyte			
Operating Systems	OS 1800 OS Irisha CP/M 3.0	OS 240	Neiron DOS1 Neiron DOS2		RAFOS	
Program Languages	B, F, P, C, Ma	F	P, A, B	B, A		
Compatibility	SM-1800 (software)	CP/M 80	MS DOS CP/M 86	Krista	Elektronika 60	Mikrosha
Applications	Education, ASNI	Field Research	Professional Work Station	School, Home	Automated Design, ASUTP, Smart Terminal	
Price (rubles)	1000					510

Name	NEVA 501	ISTRA	PK-11	SURA	UMPK-48	UMPK-80	LVOV-01
Year First Produced			1986–7	1987–8		1984–5	1987–8
Chip	K580	KR580 and	KM1801VM2	K580VM80A	KM1816VE48	KR580IK80	K580VM80A
foreign analog	Intel 8080	KR1810VM86		Intel 8080			
Speed (KOPS)							
reg-reg	400						
Word length	8	8 & 16		8	8	8	8
RAM (Kbytes)	32	1256	256	64+	64–256	2	64
ROM (Kbytes)		64	128	16+		2	16
Max. Addressable Space (Kbytes)							
Operating Systems	Internal ROM		BOS RAFOS/PK				
Program Languages			B	B			B
Compatibility			SM-4 Elektronika 60	None Known			None Known
Applications	Bookkeeping						Lab, Education, Home
Price (rubles)				995			750

Keys to Abbreviations:
Program Languages: As—ASSEMBLER; B—BASIC; C—C; F—FORTRAN; F-II—FORTRAN II; F-IV—FORTRAN IV; I—IAMB; Ma—MACROASSEMBLER; Mi—MIBOL; P—PASCAL

Sources: Artamonov (1988); Baryshnikov *et al.* (1985, 1986a, 1986b); Elorg (1986); Gorelov (1988); Kushnir *et al.* (1986); Nauka (1988b); Pogorelyi *et al.* (1986); Romanov *et al.* (1986); Tilinin (1986); Tilinin *et al.* (1986); VDNKh; Vigdorchuk *et al.* (1987a,b); Vorob'ev *et al.* (1987)

is not a "home brew" machine.[71] Like Minradioprom's ES-1841 and Minpribor's ISKRA-1031, it uses Minelektronprom's K1810VM86 copy of Intel's 8086. The machine is normally configured with 256 Kb of RAM and dual 360 Kb floppy drives. A hard disk controller is available for users lucky enough to locate the 5 MB hard disk.

The NEIRON I9.66 supports two operating systems; NEIRON-DOS1 is analogous to MS-DOS and NEIRON-DOS2 corresponds to CP/M-86. These operating systems appear to have been developed (copied) separately from similar software offered by Minradioprom and Minpribor. The manufacturer supplies several productivity software packages with the NEIRON I9.66. They include a word processor, spreadsheet, a relational data base manager, and a file manager operating under DOS1. The first three of these appear to be copies of WORDSTAR, SUPERCALC, and DBASE II.

IRISHA. The IRISHA is a "home brew" machine and a classic example of researchers and educators developing and using informal connections in the Soviet system to meet their needs for an adequate personal computer.[72] Developed by three members of the Chemistry department of Moscow State University (MSU) and with software supplied by the Moscow Institute of Informatics Problems, this machine uses the Soviet 8080 look-alike chip with up to 128 Kb memory to provide a basic educational computer. It first appeared in 1986 and is priced at 1000 rubles.

There are essentially two versions of the IRISHA, one with some form of external memory (tape recorder or floppy disk) and one diskless setup intended for network environments commonly found in Soviet classrooms. The IRISHA runs the OS 1800 operating system and, therefore, can run software written for the SM-1800 personal computer from Minpribor.

The IRISHA development story also gives interesting insight into the process by which these machines are introduced for use. Although serial production had yet to begin by 1986, considerable numbers of the IRISHA were already installed and running in the Moldavian educational system. It appears that the Central Committee of the Moldavian Communist Party, working with the designers from MSU and officials from the Academy of Sciences, managed to get enough of the machines produced with adequate software to supply some schools in the Moldavian Republic. As of 1986,

[71] Information on the NEIRON 19.66 comes from a marketing brochure published by Minpromsviazi, *Personal'naia mikro-EVM NEIRON 19.66*; Pogorelyi *et al.* (1986); and the senior author's observations.

[72] Information on the IRISHA comes from Baryshnikov *et al.* (1985); Baryshnikov *et al.* (1986a,b,c); Korneichuk and Rastorugev (1986); Romanov *et al.* (1985); and Romanov *et al.* (1986).

programmers were beginning to develop software in the Moldavian language for educational applications. One of the reasons cited for this republic's use of the IRISHA educational computer is said to be its relatively low cost.

OKEAN 240. The OKEAN 240 is another result of the centralized system's inability to supply necessary machines, thus leaving the user to meet his own demands.[73] In this case, the user is the Moscow Institute of Oceanography of the Academy of Sciences, which required a rugged and transportable microcomputer that could withstand the rigors of field and ocean research environments and would require minimal power. Based on the 8080 look-alike chip and with 128 Kb memory available, the OKEAN 240 reportedly performs 600 KOPS and uses an operating system named OS 240 that is compatible with CP/M and allows the user to attach nonstandard peripherals, presumably some types of scientific measuring equipment. One author specifically notes the machine's ability to run Microsoft FORTRAN-80.

4.2 The Academy of Sciences and Soviet "Supercomputers"

The Academy of Sciences was predominant in the field of computer design in the USSR until the mid 1960s when it went into decline. By the mid 1970s, the Academy was completely overshadowed by the computer producing ministries, and the policy of technological followership was firmly ensconced. Leadership of Lebedev's design group in the IPMCE passed to B. S. Burtsev after Lebedev's death in 1974. The Academy was short of funds and personnel. Morale was low and much of the momentum for indigenous computer development was lost.

In the 1980s, the Academy has regained some of its former luster although the power of the industrial ministries remains intact. The foci of computer design work inside the Academy in this decade, a few microcomputers such as the KORVET and OKEAN notwithstanding, has been on high-performance machines. The Soviet Union has lagged badly behind the United States, Japan, and others in the design and production of supercomputers.

Soviet scientists have finally made their political superiors aware that their nation's supercomputer gap has become a serious drag on other fields of basic and applied research. The Western policy of denying this technology to the Soviets has caused serious pain in certain technological fields, most noticeably in CAD. By 1985, Gorbachev and other top political leaders appeared to have grasped the importance of this issue.

[73] Information on the OKEAN 240 comes from Tilinin (1986); Tilinin *et al.* (1986); and Tilinin *et al.* (1987).

The 15-year plan for the development of the computer industry in the USSR, referred to earlier in this paper, features a section on the design and construction of supercomputers. In addition, the Soviet Union has prodded the members of the CMEA to produce a "coordinated" plan to begin serial production of a one billion floating-point operations per second (GFLOPS) machine by 1990 and a 10 GFLOPS machine by 1995.

4.2.1. *EL'BRUS*

In the early 1970s, Lebedev began to design a computer that would be capable of 100 MOPS, a machine that he named EL'BRUS. After he took over the IPMCE, Burtsev continued this effort.[74] From the beginning, the important design objectives were to achieve maximum integration of hardware and software design, high reliability, and very high performance. The first of this series, the EL'BRUS-1 was produced in 1978, and the improved EL'BRUS-2 appeared first in 1983. The architecture of the two machines is the same, and the superior performance of the EL'BRUS-2 is derived mainly from improved componentry.

The EL'BRUS is a modular, multi-processor, stack-based computer. It may employ as many as 10 central processors each with its associative stack memory. Basic performance data of each processor are as follows:

Fixed-point addition	520 ns.
Floating-point addition	780 ns.
Multiplication of 32-bit number	780 ns.
Multiplication of 64-bit number	1300 ns.
Logical operations	520 ns.
Million operations per second	1.5

Soviet scientists accustomed to programming the familiar BESM-6 were not particularly enthralled with the EL'BRUS-1. To attract them and to permit BESM-6 software to run on the new machine, the EL'BRUS-1 designers arranged for a special BESM-6 processor to be optionally substituted for one of the 10 regular processors. That special processor is rated at 3 MOPS. The ELBRUS-2 may not provide for the BESM-6 processor. Depending on its configuration, the EL'BRUS-1 is rated at from 1.5 to 13 MOPS; the EL'BRUS-2 is usually rated at from 10 to 100 MOPS

[74] Information on the EL'BRUS is from Burtsev (1985); Mishchenko *et al.* (1985); Artamonov (1988); and Wolcott and Goodman (1988).

although Artamonov claims that it is capable of up to 200 fixed-point operations.

The EL'BRUS-1 main memory is interleaved among from four to 32 modules of 16K 72-bit words. Memory cycle time is 1.2 microseconds. The EL'BRUS-2 can accommodate twice as much memory capacity. Both models may have up to four I/O processors each with four high-speed channel (4 million bytes/second) connections to as many as 64 ES-1066 disk drives (100 Mb each) or magnetic drum storage units. Only four of these devices may be simultaneously accessed by each I/O controller. In addition, each controller may connect with as many as 256 (16 at a time) I/O and storage devices at slower speeds (one million bytes/second).

The EL'BRUS machines may be configured with up to 16 data transmission processors, each capable of handling 160 telecommunication lines for a total of 2560. They are normally configured with RIAD peripherals.

The EL'BRUS operating system is unique in that several standard programming languages are accommodated, including FORTRAN-IV, PL/1, PASCAL, and SIMULA-67. These machines are similar in design and rated capacity to the Burroughs B-7800 which first appeared in 1979.

The EL'BRUS machines are respectable computers, at least on paper, although users have voiced the usual complaints about unsatisfactory reliability and poor peripherals. In addition, very few of the machines have been produced and that makes it very difficult for scientists to gain access to them. Development work on this line of computers continues in IPMCE and, in early 1988, Academician Velikhov reported that prototypes of an EL'BRUS-3-1 supercomputer capable of more than one billion operations per second were under construction.[75]

4.2.2. The PS-2000 and PS-3000

The PS-2000 and PS-3000 are two separate multi-processor computers that appeared in 1982. The design group included representatives from the Moscow Institute of Control Problems, the Ministry of Geology, and Minpribor which now manufactures the machines.[76]

The PS-2000 is configured with from eight to 64 8-bit processors. When fully equipped, it is said to be capable of up to 200 million fixed-point and 66.2 million floating-point operations per second. That makes it the fastest Soviet computer in serial production. Software for this machine is very limited, consisting only of a symbolic programming language and macro generator. It

[75] Velikhov (1988).

[76] Information on the PS-x000 machines is from Artamonov (1988); Mishchenko *et al.* (1985); and Wolcott and Goodman (1988).

has been manufactured in relatively small numbers and found applications in geophysics as well as elsewhere where image processing is important.

The PS-3000, which is software compatible with the PS-2000, may be equipped with up to four 32-bit scalar processors and two vector processors. Each vector processor is connected to two scalar processors. It has eight megabytes of main memory and is rated at up to eight million fixed-point scalar operations and 20 million fixed-point vector operations per second. The vector processor is said to be capable of adding 12 million additions of 32-bit floating-point numbers per second. Software for PS-3000 includes FORTRAN-II, FORTRAN-IV, ALGOL-60, and BASIC. Virtual memory, multiprogramming, and real-time processing are supported.

4.2.3. *The MARS*

A major, integrated effort to design and build a supercomputer in the 1980s was mounted by a design group called "START."[77] This effort brought the talents of IMPCE, the computer centers of Academies of Sciences in Moscow, Novosibirsk, and Tallinn together with those of Minpribor. The group's headquarters were at Akademgorodok in Novosibirsk and its purpose was to design and build the prototype of a supercomputer called "MARS" (Modular Asynchronous Expandable System).

One sub-project of the MARS effort was the design of a "mini-MARS" processor. The mini-MARS employs a modular, highly parallel, architecture. It is to be capable of 20 million floating-point operations per second (MFLOPS) with 48-bit words. Although the START group was formed only in March, 1985, the ideas behind it have been brewing with Lebedev's disciples since the late 1970s. Velikhov reported in early 1988 that the MARS design had been completed but that serial production had not yet begun. The START group has also been responsible for MARS software development.

4.2.4. *Other New Academic Computer Designs*

As in the United States, although in fewer numbers, various academic institutions in the USSR are designing computer systems reportedly capable of various levels of high performance.[78] For example, the Glushkov Institute of Cybernetics in Kiev reportedly has tested a prototype of a machine capable

[77] Information on the MARS computer is from Kotov and Marchuk (1985); and Vyshnevskii (1985).

[78] Information on these computers is from Marchuk (1987); Velikhov (1987a); Velikhov (1988); and personal interviews of the senior author.

of up to 135 MOPS. This computer is said to employ a unique, "macro-pipeline," massively parallel architecture. This machine reportedly has passed its state inspections and been approved for serial production. Many research organizations have ordered this machine but it is slated to be produced in very limited quantities.

In Leningrad, the Institute of Informatics and Automation of the Academy of Sciences developed a multi-processor system with speed to 100 MOPS. This machine also is said to have passed state inspection and been approved for serial production. The Keldysh Institute of Applied Mathematics in Moscow has cooperated with Minradioprom and Minpribor to build and operate a system said to be capable of 125 MFLOPS.

In 1988, Minelektronprom was slated to begin production of a new 32-bit microprocessor, the ELEKTRONIKA-32. The New Institute of Automatic Design reportedly has designed a new system based on this microprocessor.

Velikhov reported that the Institute of Cosmological Research together with certain Bulgarian scientists have created a parallel computer using 10 processors with "dynamic architecture and high productivity." He also said that this new machine, which is being manufactured in Bulgaria, "already enjoys popularity."[79]

In 1986, a state commission accepted the design of a supercomputer for "serial production." This machine, designed by the Academy's Institute of Cybernetics Problems and Minelektronprom, is said to be capable of 100 million operations per second. This may be the machine Velikhov called "a vector pipelined supercomputer with two levels of external memory" and may already be in limited production.[80]

Little is known about most of these various designs and prototypes. The reported association of various computer-producing ministries in these design efforts may raise the probability that some will find their way to volume production. It seems probable that many are destined to remain one-of-a-kind models. Velikhov (1988) complained that the volume of production on all Soviet high performance computers was manifestly inadequate.

One development in peripherals seems noteworthy. The Siberian Division of the Academy of Sciences together with Minelektronprom reportedly has built the prototype of a compact Winchester drive with 100 megabyte capacity.

It is clear from this brief survey that the Academy of Sciences is again very much involved in the design of high-productivity computer systems. The effort to develop and produce supercomputers is limited by several factors.

[79] Velikhov (1988), 25.
[80] *Ibid.*

Prominent among them are

- A shortage of CAD suitable for computer design.
- COCOM and other trade restrictions on technology imports.
- An underdeveloped technology base in component manufacturing.
- A semi-centralized, bureaucratic decisionmaking structure and oligopolistic industrial structure.
- A tradition of poor cooperation among production ministries and R&D organizations.
- A legacy of isolation of the Soviet computer science community from the larger world community.
- A management system that fails to reward superior performance of design and production groups, as well as to punish substandard performance.

Added to these seven factors is an eighth, namely that supercomputing was not a priority item with the Soviet leadership, at least not until 1985. The priority level has been raised in the Gorbachev regime, and computer-savvy leaders have been installed in top scientific leadership positions. Ambitious targets have been set for 1990 and 1995. Considerable progress is now being claimed, but many of the familiar ills still plague Soviet efforts to accelerate developments in this area.

4.3 Components—A Survey of Important Integrated Circuits

The Soviet Union's development of microchips and microprocessors reveals no significant exceptions to the overall pattern already seen throughout the country's entire computer development program. Much of the component base is copied from Western manufacturers, and internal organizational problems restrict the Soviet Union's efforts to develop their own technological base. Yet the desire to establish an indigenous ability to design and produce state-of-the-art micro-circuitry is complicated by the Soviet's belief that a country that finds itself technologically behind can catch up quickly by simply skipping developmental stages. It is almost as if the Soviet Union intended to design and produce industrial lathes capable of tolerances measured in millimeters, with an industrial base capable of only centimeter-level precision. They need the new machines, but the old ones cannot make them.

Soviet microchip and microprocessor technology finds itself in just such a chicken-and-egg dilemma. They want to produce 1 Mb memory chips, but have yet to ramp production of 256 Kb chips. (Their East German partners

reportedly are now beginning 1 Mb production, however.) They would like to achieve the West's 1 micron capability in integrated circuitry, but have trouble at the 3 micron level. Velikhov, the head of computing for the Academy of Sciences, states the problem this way:

> The capabilities of the organizations that design and manufacture the required technical equipment are not up to the task of the accelerated development of our microelectronics or the necessary rates of modernization. Quantitatively, we produce about 10% of the output of analogous equipment in the West. Qualitatively, we lag significantly behind foreign producers. But given that we have little equipment, we are unable to design new generations of ICs. As a result, in both logical design and "memory"—the most fundamental work—we lag by two generations. At present, we have a dearth of 64 Kb memory media while abroad, they are beginning to sell megabyte media more than ten times cheaper.[81]

The following brief survey of Soviet microprocessors and memory chips will reveal how much they have relied on technology followership. During the remainder of this century, the only way for the Soviet Union to catch up to current Western state-of-the-art standards would be to import Western designs and manufacturing machinery. The likelihood of that, even with more *perestroika* and *glasnost'*, remains slim. Thus, Soviet development in the near term seems dependent on how much they can garner from their East German partners.

4.3.1. Soviet Microprocessor Chips

Any review of Soviet capabilities in microprocessor technology must begin with a note about the availability of source materials. First and foremost, the Soviet Union does not publish production figures of these components. Because Minelektronprom, the monopoly producer of Soviet microprocessors, is a VPK ministry, its production activities are difficult to track down precisely. Despite this lack of production information, however, the numerous journals and books that are available do indicate the sources, uses, and capabilities of Soviet microprocessors.

Table XIII outlines the main processors identified with applications in Soviet computing. The list is not an exhaustive treatment of all Soviet processors. It does reveal a traditional, and continued, reliance on bit-slice technology, which undoubtedly is a reflection of Soviet manufacturing capabilities. It also reveals limited design capabilities that, undoubtedly, stem

[81] Velikhov (1987b), 23.

TABLE XIII

SAMPLE LIST OF SOVIET MICROPROCESSORS

Chip Series	K589	K1802	K1804	K1800	K587	K588	K536	K583	K584
Processor Chip	K589IK02	KR1802VS1*	KR1804VS1	K1800VS1*	K587IK2	KR588VS2*		K583IK3	KR584IKIA
Foreign Analog	Intel 3000	NK	AM 2900	M10800	NK	NK	NK	NK	SBP 0400
Manuf. Tech.	TTLS	TTLS	TTLS	ECL	CMOS	CMOS	PMOS	IIL	IIL
Chip Type	Bit-slice	Bit-slice	Bit-slice	Bit-slice	Bit-slice	Bit-slice	Bit-slice	Bit-slice	Bit-slice
Word Length (bits)	2	8	4	4	4	16	8	8	4
Cycle Time (msec)	0.1	0.15	0.12	0.04	2	2–5	10	1	2
Clock Speed (MHz)	6	8	8	36	0.5	1		1	0.5
Production Began	late 1970s		early 1980s	early 1980s			mid 1970s		late 1970s
Applications	ES, SM, ISKRA 555, T3-29MK	SM-1300, ES	Elek 80-1, SM, ES	ES	NTs series	AGAT, PK80xx, NTs series	S5 micros	ES	Calculators

Chip Series	K581	K586	K1811	K1801	K580	K1810
Processor Chip	KR581VE1	K586IK1	KN1811VM1*	K1801VM1	KR580IK80A	K1810VM8
Foreign Analog	LSI-11/2 T1602	NK	NK	NK	Intel 8080	Intel 8086
Manuf. Tech.	NMOS	NMOS	NMOS	NMOS	NMOS	NMOS
Chip Type	Multichip	Multichip	Multichip	Single	Single	Single

Word Length (bits)	16	16	16	16	8	16
Cycle Time (msec)	0.4	0.5		2	2	1
Clock Speed (MHz)	2.5–3.3			8		4–5
Production Began	late 1970s				1979	mid 1980s
Applications	ELEK. 60, 60M, KVANT, SM	S5-21	ELEK 60-1	S5-41, BK-0010, DVK series, NTs series	SM-1800, OKEAN 240, KORVET, et al	ES-1840, NEIRON, SM-1810, PK-11, ISTRA ISKRA 255

Keys to Abbreviations:
*—Arithmetic-Logic Unit
NK—None Known or Cited in Soviet Sources
Chip Type: Multichip—Multichip Microprocessor
Single—Singlechip Microprocessor
Manuf. Tech.: CMOS—Complementary Metal-oxide Semiconductors
ECL—Emitter-Coupled Logic
IIL—Integrated Injection Logic
NMOS—*n*-channel Metal-oxide Semiconductor
PMOS—*p*-channel Metal-oxide Semiconductor
TTLS—Transistor-Transitor Logic with Schottky

Sources: Dshkhunian *et al.* (1984); Faizulaev and Tarabrina (1986); Govorun *et al.* (1986); Grishin and Ugol'kov (1985); Heuertz (1983, 1984); Iakubovskii *et al.* (1984); Ivanov *et al.* (1986); Khvoshch *et al.* (1985); Kobylinskii *et al.* (1986); Korneichuk and Rastorguev (1986); Kuleshova (1987); Lopatin *et al.* (1985); Luk'ianov (1985); Malashevich (1984); Nesterov *et al.* (1986); Presnukhin (1986a,b); Proleiko (1984); Shaknov (1984); Solov'ev (1985); Stapleton (1985).

from a lack of CAD. It is not without reason that the Soviets decided to copy the Intel 8080 and Intel 8086 microprocessor chips. In addition, it is not surprising to note the absence of an 80286 or 80386 analog. That level of design and manufacturing ability remains out of reach. As one Soviet author writes, "At this time there is no domestic analog to [the 80286] microprocessor and it is not visible 'on the horizon.'"[82]

4.3.2. *Soviet Memory Chips*

It is difficult to compare Soviet memory chips to present Western chips because the West continues to develop so quickly. Suffice it to say that at this point the West is producing 1 Mb dynamic random access memory (DRAM) chips and is about to begin production of 4 Mb DRAM chips. The Soviet

TABLE XIV

SOVIET MEMORY CHIPS

Static RAMs			
Type	Man. tech.	Capacity (bits)	Access time (ns)
KR188RU2A	CMOS	256	500
564RU2	CMOS	256	650
505RU4	pMOS	256	850
132RU4	nMOS	1024	25
KR565RU2A	nMOS	1024	450
KR185RU5	TTL	1024	330
500RU415	ECL	1024	30
KR537RU2A	CMOS	4096	300
KR541RU1A	IIL	4096	120
KR541RU3	IIL	16384	100
KR132RU6A, B	nMOS	16384	?
Dynamic RAMs			
Type	Man. tech.	Capacity (bits)	Access time (ns)
KR507RU1	pMOS	1024	400
KR565RU1A	nMOS	4096	150–200
K565RU3A	nMOS	16384	150–200
K565RU5B	nMOS	64K	?

Sources: Baranov *et al.* (1986); Iakubovskii *et al.* (1984); Solov'ev (1985).

[82] Shirokov (1988), 43.

Union, on the other hand, can competently produce 16K chips, only recently began volume production of 64K chips, and, while it has prototypes, has yet to produce 256K chips in quantity. In short, there is no contest in this area.[83] Table XIV gives a sample of current Soviet memory chips.

4.3.3. *General-Purpose Integrated Circuits*

The Soviet Union's decision to copy Western component designs is revealed most starkly by a quick survey of general-purpose integrated circuits. For each type of manufacturing process, Soviet computer engineers copied a specific Western series. In transistor-transistor logic, the SN54/74 series served, and serves, as the model. The MC10000 series of ICs provided the Soviets with the model for Emitter-Coupled Logic chips. And finally, in the CMOS area, the RCA CD4000 series served as the example to copy.

5. Perestroika and Soviet Computing

The fragmented, isolated, and ill-managed Soviet computer development effort not only proved incapable of keeping pace with Western developments, but also failed to fulfill its own goals. It was evident to Western analysts many years ago that the bureaucratic structure of Soviet science and technology severely restricted technological development. The question now is whether the Soviet leadership recognizes the same, and is able to do anything about it. A first glance at the effects of *perestroika*, or economic restructuring, on the computer industry would give much hope for improvement. A deeper view, however, reveals that much of what *perestroika* has achieved so far amounts simply to bureaucratic reshuffling. Soviet insistence on centrally managed development, coupled with their inability to do it, continues to impede the development of the Soviet computer industry to the end of the 1980s.

5.1 The Reemergence of the Academy of Sciences

Essentially shut out from general developments in Soviet computing since the mid 1960s, the Academy of Sciences undertook a significant organizational restructuring in its computer development program in 1983, including the establishment of a new Department of Informatics, Computer Technology and Automation (OIVTA) as well as a series of new research institutes. The developer of the SM minicomputer line and advocate of the

[83] Information on Soviet memory chips is from Iakubovskii *et al.* (1984); Solov'ev (1985), 240–241; and Baranov (1986), 357–358.

Academy's reemergence into computer developments, B. N. Naumov, stated, "In order not to repeat these mistakes [of leaving the Academy out of computers], the newly created Department of Informatics, Computer Technology, and Automation of the USSR Academy of Sciences should take upon itself the leading role in the design and implementation of a unified scientific-technological policy in this area."[84]

The Academy is now involved in a full range of computer development programs, including the EL'BRUS supercomputer, mini- and microcomputers, and computer chip manufacturing processes. One Western analyst notes that this reorganization "... means that the control over a substantial portion of the research-production process in the computer field now passes from industry to the Academy."[85] While it is quite clear Velikhov and colleagues have brought the Academy back into the computer field, whether it is *the* leader is yet to be seen.

5.2 A New "Tsar" for Soviet Computing?

Our skepticism is based on additional administrative restructuring that occurred after the Academy reorganization supposedly placed it at the head of technological development. Created March 21, 1986 by the Presidium of the Supreme Soviet, the USSR State Committee for Computer Technology and Informatics (GKVTI) is supposed to coordinate the creation, production, utilization, and servicing of computer technology. The new committee chairman, Nikolai Vasil'evich Gorshkov, who was appointed April 7, 1986,[86] stated that the "new committee effectively is the lead organization of the powerful interdepartmental scientific technological complex (MNTK) for the development, production, and implementation of the means of computer technology."[87] Despite the resolution's and Gorshkov's statements, however, it remains doubtful as to whether the GKVTI is or will be *the* coordinating entity of the Soviet computer industry.

The first indication of resistance to GKVTI coordination is simply the amount of time from the formation of the committee, March 1986, to the ratification of the Council of Ministers' statute in April 1987. In an April 1986 interview, Gorshkov stated that "we must ... work out and determine the organizational documents in a three-month term."[88] The fact that it took more than a year indicates that considerable resistance was encountered.

[84] Naumov (1984).
[85] Kassel (1986), *vi.*
[86] *Pravda*, April 8, 1986.
[87] *Ekonomicheskaia Gazeta*, (18), 1986.
[88] *Ibid*

The Council of Ministers statute assigns a great deal of responsibility to the GKVTI, and at least on paper, considerable power.[89] But those institutional powers conflict with the interests of existing organs, especially the powerful manufacturing ministries. In the area of planning, the GKVTI is responsible for determining the basic directions, priorities, future demands, etc., for computer technology in the Soviet Union. The GKVTI, according to the statute, prepares proposals for annual and five-year plans. It remains unclear how this fits into other reports that Velikhov drafted the long-term computer development plan from his base at the Academy of Sciences.

The Academy–Industry confusion and conflict is also apparent in the leadership of the GKVTI. Gorshkov, the Committee's chairman, is a trained engineer and served in Minradioprom management since 1964. He was a Deputy Minister for Minradioprom beginning in 1974.[90] Given this background and the existing feud, it appears that the GKVTI might tend to represent industrial interests in the turf wars.

On paper, the GKVTI is a powerful organ. The statute gives arbitration rights to the GKVTI for settling interdepartmental quarrels, and it allows the GKVTI to set out "binding" resolutions on ministries, departments, enterprises, institutions, and organizations "within its jurisdiction." The GKVTI "can create, reorganize, and abolish enterprises, institutions, and organizations within its competency." But some of the inconsistencies noted above, along with the historical record of past failed attempts to do similar things, leaves significant doubts as to whether the GKVTI is the bureaucratic entity that will coordinate Soviet computer development. Much as the GKNT remained a rather feeble centralizing and coordinating body for all of Soviet science and technology, the new GKVTI appears to suffer from the same maladies.

5.3 Bureaucratic Shuffling Continues

The situation becomes even more confused with the introduction of a new bureaucratic entity in Soviet technological development, the MNTK. MNTKs are designed to provide administrative flexibility and a connection between research, development, and production by including appropriate academic institutes, scientific-production associations, enterprises, and ministries in consortiums to tackle specific problems. Since 1985, more than 20 MNTKs have been formed to coordinate efforts in developing personal computers, robots, biogenetics, fiber optics, cotton harvesters, etc. As with the GKVTI, MNTKs are afforded a fair amount of power on paper, but in

[89] *Sobranie Postanovlenii Pravitel'stva SSSR Pervyi Otdel*, 596–603.

[90] Kassel (1986), 26.

practice have been unable to improve the situation. Naumov, Director of the MNTK for personal computers, noted in a letter to the editor in Pravda,

> It is approximately two-and-a-half years since the decision to create intersector scientific and technical complexes was adopted, but there is still no economic mechanism for their work, and an experimental and experimental-production base has not been created.[91]

Thus, this most recent attempt to improve upon the centralized management of technological development fails to improve the research-design-production connections.

One of the most novel administrative entities to appear in Soviet technological development is the Temporary Scientific Collective (TSC). Restricted to a life of three years by Council of Minister statute, this type of organization is intended to solve a specific problem in a short time, and then dissolve itself when other priorities come to the fore. As of March 1987, approximately 10 TSCs were in operation.

Created in 1985, the START Collective was one of the first to be organized and served as a test case for the entire concept. Its role in developing the MARS computer was described in Section 4.2.3, above. Now out of existence, it employed a total of 155 people, including researchers from the computer center, the Cybernetics Institute of the Estonian Academy of Sciences, and specialists from the Severodonetsk Impuls Scientific Production Association of Minpribor.

Yet, even this most flexible organizational entity has encountered problems. As with the MNTKs, the inter-departmental nature of the TSC leaves it bureaucratically "orphaned." Supplies are difficult to acquire. Office space is hard to find. Even housing for staff can be a major headache. The director of START, V. E. Kotov, admits breaking administrative statutes by hiring managerial staff under phony job descriptions in order to get the necessary work done. Although collectives have the right to hire short-term workers, they often lack the funds to do so. Thus, the TSC collides with the very problems it is designed to overcome.

5.4 Calls for More of the Same

The organizational problems within the Soviet computer industry were recently discussed by the Supreme Soviet Standing Commission on Science and Technology, the Soviet equivalent of a parliamentary committee on

[91] *Pravda*, May 4, 1988, 3.

science policy.[92] The committee discussed delays in both plant development and serial production of computers, the inability to achieve world standards, the continued production of non-compatible PCs, and the poor level of computer training. A TASS report on the committee meeting stated,

> The calamity is that our planning organs cannot part at all from gross indicators. Financially, our plans for electronics are being fulfilled, in a manner of speaking, but in terms of the range of goods, it is a total catastrophe.[93]

The committee then cited organizational problems as the cause of this "total catastrophe." First and foremost, they note that the GKVTI lacks the financial levers with which to manage the computer industry. The ministries continue to hold the purse strings, leaving these new bureaucratic entities stranded unless they can attach themselves to a ministry. The Supreme Soviet committee's response to the problem, however, is to try more of the same centralized effort that has failed for the last 20 years. The report states,

> Obviously, it is expedient to examine the issue of setting up a national economic complex that will unite the sectors and implement a unified scientific and technical policy.[94]

They recognize a problem, but always come back to the same solution, a new and improved centralized effort. When will they learn?

Summary

The Soviet policy of copying Western hardware design, combined with international isolation and an industrial structure that retards domestic development, production, and support, effectively doomed Soviet computerdom to an expanding lag behind the West during the 1980s. The strategy of technological followership, by itself, did not cause the problem. After all, many other countries successfully rode the high-tech wave of the 1980s while pursuing such a strategy. Their success, however, was due to their integration into the world economy and the necessity of their producers to compete in open markets. Isolated from the stimulation of such competition, Soviet computer development wallowed in a "hot house" domestic economy that purposely shunned competition in favor of centralized bureaucratic control.

[92] See Ivakhnov (1988).

[93] Deputies See 'Critical' Science, Technology Lag. Moscow Radio, August 15, 1988, as reported by FBIS-SOV-88-158, 63.

[94] *Ibid.*

In summary, there is both good news and bad news for the Soviet computer user of the late 1980s. The bad news is that available hardware and software continues to fall farther behind what their Western counterparts are using at every level, from supercomputers to microcomputers. The good news is that the Soviet's scientific and political leadership now openly recognizes the problem and vows to resolve it. But it seems doubtful that the attempted solutions, which so far have been limited to industrial reorganization and bureaucratic reshuffling, will be sufficient to slow the rate at which they are falling behind, much less actually begin to close the gap. Unfortunately, for the Soviet computing community, the headlines for the 1990s will probably read: Soviet computer industry continues to lag; Soviet leadership continues to do something about it.

References

Abramovich, S. N., *et al.* (1985). Professional'nye Personal'nye EVM "Iskra 226." *Mikroprotsessornye Sredstva i Sistemy* (2), 29–36.

Adamovich, A. I., and Leonas, V. V. (1988). Virtual'nye Diski v MNOS. *Programmirovanie* (3), 92–94.

Artamonov, G. T., ed. (1988). "Sredstva Informatsionnoi Tekhniki." Energoatomizdat, Moscow.

Ashastin, R. (1980). Sotrudnichestvo v Oblasti Vychislitel'noi Tekhniki na Pod'eme. *Ekonomicheskoe Sotrudnichestvo Stran-chlenov SEV* (3), 80–84.

Baranov, V. V., *et al.* (1986). *Poluprovodnikovye BIS Zapominaiushchikh Ustroistv: Spravochnik.* Radio i Sviaz', Moscow.

Baryshnikov, V. N., *et al.* (1985). Personal'naia EVM "Irisha" Dlia Kabinetov Informatiki i Vychislitel'noi Tekhniki. *Mikroprotsessornye Sredstva i Sistemy* (3), 5–8.

Baryshnikov, V. N., *et al.* (1986a). Modul' Protsessora Personal'noi EVM "Irisha". *Mikroprotsessornye Sredstva i Sistemy* (2), 52–62.

Baryshnikov, V. N., *et al.* (1986b). Programmnoe Obespechenie PEVM "Irisha". *Mikroprotsessornye Sredstva i Sistemy* (3), 59–64.

Baryshnikov, V. N., *et al.* (1986c). Programma Vyvoda Simvol'noi i Graficheskoi Informatsii Personal'noi EVM "Irisha". *Mikroprotsessornye Sredstva i Sistemy* (4), 79–89.

Basin, A. (1988). "Agat" v nashei shkole. *Informatika i Obrazovanie* (3), 98–100.

Birbilas, A. Iu., and Strishka, V. Ch. (1988). Avtomatizatsiia Podgotovitel'nykh Operatsii v Proizvodstve Magnitnykh Diskov. *Pribory i Sistemy Upravleniia* (3), 35–37.

Boiko, V. V., and Savinkov, V. M. (1987). Sravnitel'nyi Analiz Promyshlennykh SUBD. *Prikladnaia Informatika* **1** (12).

Bores, L. D. (1984). Agat: a Soviet Apple II Computer. *BYTE* (11), 135–136, 486–490.

Briabrin, V. M., and Chizhov, A. A. (1986). Arkhitektura Operatsionnoi Sistemy Al'fa-DOS/DOS-16. *Mikroprotsessornye Sredstva i Sistemy* (4), 51–56.

Broczko, P. (1985). The Microcomputer Manufacture of Socialist Countries in 1984. *Szamitastechnika* (4), 8–9.

Broczko, P. (1988). A Szocialista Orszagok Mikroszamitogepgyartasa. *Szamitastechnika* (Hungarian), May 15, 12–14.

Burtsev, V. S. (1985). Nauchnoe Nasledie Akademika S. A. Lebedeva. *Kibernetika i Vychislitel'naia Tekhnika* **1**, 5–10.

Buzin, A. (1988). Agat v Nashei Shkole. *Informatika i Obrazovanie* (3), 98–100.

Campbell, H. (1976). Organization of Research, Development and Production in the Soviet Computer Industry. RAND Corporation, R-1617-PR, Santa Monica, California.

Computer Review 1987. **27** (2), GML Corporation.

Dale, A. G. (1979). Database Management Systems Development in the USSR. *Computer Surveys* **11** (3), 213–226.

Data Decisions: Computer Systems (July 1983) Business Publications Division, Ziff-Davis Publishing Company, New York.

Davis, N. C., and Goodman, S. E. (1978). The Soviet Bloc's Unified System of Computers. *Computing Surveys* **10** (2), 93–122.

Denisenko, A. (1986). Ispytaniia KUVT-86. *Informatika i Obrazovanie* (2), 69–74.

"Deputies See 'Critical' Science," Technology Lag. Moscow Domestic Service, August 15, 1988, as reported by FBIS-SOV-88-158, 63.

Driga, I. (1986). Oborudovanie dlia KVT. *Informatika i Obrazovanie* (2), 66–69.

Dshkhunian, V. L., *et al* (1984). Odnokristal'nye Mikroprotsessory Komplekta BIS Serii K1801. *Mikroprotsessornye Sredstva i Sistemy* (4), 12–18.

Dshkhunian, V. L., *et al.* (1985). Odnoplatnye MikroEVM Riada "Elektronika MS 1201." *Mikroprotsessornye Sredstva i Sistemy* (2), 8–13.

Dujnic, P., and Fundarek, M. (1983). Automation trends in management. *Informacne Systemy* (4), 302–394.

Elektronno-vychislitel'naia mashina ES 1007. (1988) (brochure) Minradioprom.

Elorg Information (1986). (1), Vneshtorgreklama.

Elorg Informiruet (1983). (6), 38. Vneshtorgreklama.

Ershov, A. P. (1975). A History of Computing in the U.S.S.R. *Datamation* **21** (6), 80–88.

Ershov, A. P. (1986). 1,100,000 PEVM: vovremia podgotovit'sia, nichego ne upustit' iz vidu. *Mikroprotsessornye Sredstva i Sistemy* (4), 2.

Ershov, A. P. (1987). Shkol'naia informatika v SSSR: ot gramotnosti k kulture. *Informatika i Obrazovanie* (6), 3–11.

Faizulaev, B. N., and Tarabrina, B. V. (1986). "Spravochnik: Primenenie Integral'nykh Mikroskhem v Elektronnoi Vychislitel'noi Tekhnike." Radio i Sviaz', Moscow.

Filinova, E. N. (1985). "Diskovaia Operatsionnaia Sistema Kollektivnogo Pol'sovaniia dlia SM EVM." Finansy i Statistika, Moscow.

Fortunov, I. (1985). Programmnaia Sistema dlia Obrabotki Tekstov TOS-83 na Mini-EVM Tipa SM-4. *Vychislitel'naia Tekhnika Sotsialisticheskikh Stran* **17**, 68–72.

Giglavyi, A. V., *et al.* (1984). "MikroEVM SM-1800." Finansy i Statistika, Moscow.

Glushkov, V. M. (1979). Vychislitel'naia Tekhnika v SSSR. *In* "Kibernetika: Voprosy Teorii i Praktiki" (V. M. Glushkov), pp. 219–228. Nauka, Moscow.

Glushkova, G. G., and Ivanov, E. A. (1986). MikroEVM Semeistva "Elektronika." *Mikroprotsessornye Sredstva i Sistemy* (4), 7–11.

Goodman, S. E. (1979). Software in the Soviet Union: Progress and Problems. *Advances in Computers* **18**, 231–287.

Goodman, S. E., *et al.* (1984). General-Purpose Computer Systems in the Warsaw Pact Countries. *Signal*, December, 97–101.

Gorelov, S. (1988). Bytovye PEVM Stanoviatsia Blizhe. *Radio* (8), 62.

Goskomstata SSSR (1988). "Paket Programm DIALOG na PEVM." (brochure), VNIPIstat-inform Goskomstata SSSR, Moscow, 1/20/88.

Govorun, V. N., *et al.* (1986). MikroEVM i Upravliaiushchie Moduli na Baze BIS Serii K1810. *Mikroprotsessornye Sredstva i Sistemy* (5), 13–15.

Grevtsev, V. V. (1988). Sredstva peredachi dannykh mikroEVM semeistva SM 1800. *Mikroprotsessornye Sredstva i Sistemy* (2), 41–43.

Grif, A. (1988). "Korvet" na meli, kto vinovat? *Radio* (7), 2–4.

Grigor'ev, A. G. (1987). Adaptirovannaia Operatsionnaia Sistema ADOS dlia SM EVM. *Mikroprotsessornye Sredstva i Sistemy* (4), 23–25.

Grishin, V. A., and Ugol'kov, V. N. (1985). "Sektsionnye Mikroprotsessory i Ikh Programmirovanie." Nauka, Novosibirsk.

Heuertz, R. (1983). "Microcomputer Development in the U.S.S.R." MA Thesis, University of Kansas. University Microfilms International, Ann Arbor, Michigan.

Heuertz, R. (1984). Soviet Microprocessors and Microcomputers. *Byte*, April, 351–362.

Iakubaitis, E. A. (1985). "Lokal'nye Informatsionno-Vychislitel'nye Seti." Zinatne, Riga.

Iakubovskii, S. V., *et al.* (1984). "Analogovye i Tsifrovye Integralnye Mikroshkemy." Radio i Sviaz, Moscow.

Iaroshevskaia, M. B. (1986). Personal'naia EVM "Iskra" 1030.11. *Mikroprotsessornye Sredstva i Sistemy* (4), 23–24.

Ioffe, A. G. (1984). Massovye personal'nye EVM serii "Agat." *Mikroprotsessornye Sredstva i Sistemy* (1), 56–60.

Isaev, M. A. (1987). E-Praktikum dlia ES EVM. *Mikroprotsessornye Sredstva i Sistemy* (3), 76.

Ivakhnov, A. (1988). Elektronika: Vdogonku za Bcherashnim Dnem. *Izvestiia*, August 15, 3.

Ivanov, G. (1988). Tekstovye Protsessory. *Radio* (7), 26–28.

Ivanov, S. N., *et al.* (1986). Odnoplatnaia MikroEVM na MPK BIS Serii K1810. *Mikroprotsessornye Sredstva i Sistemy* (6), 8–13.

Ivanova, S. B. (1987). Vsesoiuznyi nauchno-tekhnicheskii seminar. *Pribory i Sistemy Upravleniia* (12), 40–41.

JS (1984). JSEP 3—Next Stage of Development of the Uniform System of Electronic Computers. *Vyber Informacz z Organizachi a Vypocetni Techniky* (in Czech) (2), 195–196.

Judy, R. W. (1967). Information, Control, and Soviet Economic Management (and Appendix: Characteristics of Some Contemporary Soviet Computers). *In* "Mathematics and Computers in Soviet Economic Planning" (J. P. Hardt, M. Hoffenberg, N. Kaplan, and H. S. Levine, eds.), pp. 1–67 (and 261–265). Yale University Press, New Haven.

Judy, R. W. (1970). The Case of Computer Technology. *In* "East-West Trade and the Technology Gap" (Stanislaw Wasowski, ed.), pp. 43–72. Praeger Publishers, New York.

Judy, R. W. (1986). The Riad Computers of the Soviet Union and Eastern Europe, 1970–1985: A Survey and Comparative Analysis. HI-3872, Hudson Institute, Indianapolis.

Jungnickel, H. G. (1984). The work at the ESER. *Rechentechnik-dataverarbeitung* **21**, (10), 5–8.

Kabelevskii, A. N. (1986). "Malye EVM: Funktsional'noe Proektirovanie." Nauka, Moscow.

Kaloshkin, E. P., *et al.* (1985). Mikroprotsessornyi Komplekt BIS Serii K583. *Mikroprotsessornye Sredstva i Sistemy* (2), 18–23.

Karpilovich, Yu. V. (1983). YeS-1061 Is on Good Path. *Sovetskaia Belorussiia*, 3 April, 1.

Karrasko, L. Kh. (1980). Razvitie Malykh EVM Na Kube. *Vychislitel'naia Tekhnika Sotsialisticheskikh Stran* (8), 19–23.

Kassel, Simon (1986). A New Force in the Soviet Computer Industry: The Reorganization of the USSR Academy of Sciences in the Computer Field, Rand Note N-2486-ARPA.

Kezling, G. B., ed. (1986). "Tekhnicheskie Sredstva ASU: Spravochnik 2." Mashinostroenie, Leningrad.

Khanov, M., and Eremin, A. (1983). Minicomputers in Automated Control Systems: Elektronika-100-25. *Tekhnika i Vooruzhenie* (5), 4–5.

Khatskevich, L. D., and Protsenko, I. G. (1988). Professional'naia Personal'naia EVM "Elektronika MS 0585." *Mikroprotsessornye Sredstva i Sistemy* (2), 3–6.

Khvoshch, S. T., *et al.* (1985). "Inzhektsionnye Mikroprotsessory v Upravlenii Promyshlennym Oborudovaniem." Mashinostroenie, Leningrad.

Kobylinskii, A. V., *et al.* (1986). Odnokristal'nyi Vysokoproisvoditel'nyi 16-razriadnyi Mikroprotsessor KM1810VM86. *Mikroprotsessornye Sredstva i Sistemy* (1), 28–33.

Koisina, L. (1988). Otkrytoe pis'mo direktory zavoda "Kvant." *Informatika i Obrazovanie* (5), 118.

Kokorin, V. S., *et al.* (1986). Tendentsiia Razvitiia Dialogovykh Vychislitel'nykh Kompleksov. *Mikroprotsessornye Sredstva i Sistemy* (4), 11–15.

Korneichuk, A. A., and Rastorguev, A. A. (1986). Kruglyi Stol s Ostrymi Uglami. *Mikroprotsessornye Sredstva i Sistemy* (2), 92–93.

Korneichuk, V. I., *et al.* (1986). "Vychislitel'nyie Ustroistva na Mikro-Skhemakh: Spravochnik." Tekhnika, Kiev.

Korolev, L. N., and Mel'nikov, V. A. (1976). Vychislitel'nye Sredstva i Vspomogatel'noe Oborudovanie Sistem. *Upravliaiushchie Sistemy i Mashiny* (6), 7–11.

Kostelianskii, V. M., and Resanov, V. V. (1980). Upravliaiushchie Vychislitel'nye Kompleksy SM-1 i SM-2. *Vychislitel'naia Tekhnika Sotsialisticheskikh Stran* (8), 30–38.

Kotov, V. E., and Marchuk, A. G. (1985). Proekt MARS—Kompleksnyi podkhod k rasrabotke vychislitel'nykh sistem. *In* "Kibernetika i vychislitel'naia tekhnika" (A. Mel'nikov, ed.) (1), pp. 69–77. Nauka, Moscow.

Kozirev, S., and Sokolov, S. (1987). Computer with an Accent: Why are They Going to the Black Market for Personal Computer Software and Not to Stores? *Komsomolskaia Pravda*, March 3, 2, as translated in JPRS-UCC-87-018, September 28, 1987, 8–11.

Krilov, V. V. (1985). Nauka o programmakh. *Mikroprotsessornye Sredstva i Sistemy* (1), 42–43.

Kuchukian, A. T., Sarkisian, T. E., and Ter-Israelian, V. A. (1985). ES-1046-EVM vysokoi proizvoitel'nosti. *Vychislitel'naia Tekhnika Sotsialisticheskikh Stran* **18,** 174–179.

Kuleshova, V. I. (1987). Mikroprotsessornyi Komplekt Serii KR580. *Mikroprotsessornye Sredstva i Sistemy* (5), 87–94.

Kushnir, V. E., *et al.* (1986). Uchebnaia MikroEVM na Osnove Odnokristal'noi EVM KM 1816VE48. *Mikroprotsessornye Sredstva i Sistemy* (6), 75–82.

Kuznetsov, C. O., Lanko, A. A., Leont'ev, D. I., Matveev, O. B., Prokhorov, N. L., Raev, V. K., and Shotov, A. E. (1988). Elektronnyi Disk SM 5803 Dlia MikroEVM S Interfeisom "Obshchaia Shina." *Mikroprotsessornye Sredstva i Sistemy* (2), 78–81.

Larionov, A. M. (1976). Edinaia sistema EVM i perspektivy ee razvitiia. *Voprosy Kibernetiki* **20,** 61–74.

Larionov, A. M. (1977). Osnovnye kontseptsii razvitiia ES EVM. *Vychislitel'naia Tekhnika Sotsialisticheskikh Stran* **1,** 41–51.

Larionov, A. M., Levin, V. K., Przhiialkovskii, V. V., and Fateev, A. E. (1973) Osnovye printsipy postroeniia i tekhniko-ekonomicheskie kharakteristiki Edinoi Sistemy EVM (ES EVM). *Upravliaiushchie Sistemy i Mashiny* (3), 1–12.

Lavreniuk, Iu. A., Belynskii, V. V., Golubev, B. P., and Zenin, V. M. (1979). Pervye sovmestnye ispytaniia tekhnicheskikh i programmykh sredstv SM EVM. *Vychislitel'naia Tekhnika Sotsialisticheskikh Stran* **5,** 117–122.

Legavko, A. V., and Vasilenko, S. N. (1988). Organizatsiia mezhmashinnoi sviazi mezhdu mikroEVM "Elektronika 60M" i mini-EVM SM-1. *Pribory i Sistemy Upravleniia* (4), 31.

Lemko, L. M., *et al.* (1987). Personal'nyi Mikrokomp'iuter "Elektronika MK 85." *Mikroprotsessornye Sredstva i Sistemy* (4), 10–12.

Levnina, G. A. (1988). Vychislitel'nyi Kompleks SM 1700: Arkhitektura i Tekhnicheskie Sredstva. *Pribory i Sistemy Upravleniia* (3), 1.

Lipaev, V. V., *et al.* (1985). Sistema Avtomatizatsii Proektirovaniia Programm na Baze Personal' nykh EVM (Sistema PRA). *Mikroprotsessornye Sredstva i Sistemy* (4), 42–45.

Loeschner, V., and Kasper, B. (1984). Computer equipment at the 1984 Leipzig Spring Fair. *Radio Fernsehen Elektronik* **33** (7), 430–463.

Lomov, Iu. S. (1987). EVM vuysoiki proizvoditel'nosti ES-1066 i ES-1065. *In* "Elektronnaia Vychislitel'naia Tekhnika" 1 (V. V. Przhiialkovskii, ed.), pp. 177–188. Radio i Sviaz', Moscow.

Lopatin, V. S., *et al.* (1985). MikroEVM "Elektronika MS 1211," "Elektronika MS 1212." *Mikroprotsessornye Sredstva i Sistemy* (2), 14–15.

Lopato, G. P., Smirnov, G. D., and Pykhtin, V. Ia. (1986). Sovetskie personal'nye professional'nye EVM edinoi sistemy. *Vychislitel'naia Tekhnika Sotsialisticheskikh Stran* **20,** 3–11.

Luk'ianov, D. A. (1985). KR580—Avtomatizatsiia Bez Problem! *Mikroprotsessornye Sredstva i Sistemy* (1), 85–90.

Malashevich, B. M. (1984). Magistral'no-Modul'nye Mikroprotsessornye Sistemy. *Mikroprotsessornye Sredstva i Sistemy* (4), 3–11.

Maliarskii, N. M., and Terekhov, Iu. V. (1987). Mikroelektronnaia elementnaia basa sredstv vychislitel'noi tekhniki. *In* "Elektronnaia Vychislitel'naia Tekhnika" **1** (V. V. Przhiialkovskii, ed.), pp. 210–216. Radio i Sviaz', Moscow.

Marchuk, G. I. (1986). "Vychislitel'nye protsessy i sistemy" **2.** Nauka, Moscow.

Marchuk, G. I. (1987). Vystupitel'noe slovo presidenta Akademii Nauk SSSR. *Vestnik Akademii Nauk* (7), 7–18.

Marples, D. (1985). The Computer in the Ukraine: Some New Developments. Radio Liberty Research Report 179/85, May 30, 1985.

Mel'nikov, V. A. (1986). S. A. Lebedev—osnovopolozhnik otechestvennoi vychislitel'noi tekhniki. *Informatika i Obrazovanie* (1), 6–13.

Mikrodos. (1985). MIKRODOS—Mobil'naia Operatsionnaia Sistema dlia MikroEVM. *Mikroprotsessornye Sredstva i Sistemy* (2), 92.

Mishchenko, V. A., Lazarevich, E. G., and Aksenov, A. I. (1985). "Raschet proizvoditel'nosti mnogoprotsessornykh vychislitel'nykh sistem." Vyssheishaia Shkola, Minsk.

Murenko, L. L., *et al.* (1986). Personal'naia EVM "Elektronika T3-29MK." *Mikroprotsessornye Sredstva i Sistemy* (4), 20–23.

Musaelian, V. (1985). *Kommunist*, 7 July, 1.

Muzychkin, P. A., *et al.* (1988). Avtomatizirovannaia Obychaiushchaia Sistema dlia EVM Personal'nogo Uspol'zovania. *Programmirovanie* (3), 70–80.

Nanassy, T. (1985). A comparison of the ES 1034 and 1032 computers. *Szamitastechnika*, March 4.

Nauka. (1988a). Komp'iuternyi Klass k Pod" ezdu Podan. *Nauka i Zhizn'* (4), 13.

Nauka. (1988b). Ne Tol'ko BK. *Nauka i Zhizn'* (4), 123–124.

Naumov, B. N. (1977). Sozdanie SM EVM—Novyi Etap Razvitiia Sredstv Vychislitel'noi Tekhniki. *Vychislitel'naia Tekhnika Sotsialisticheskikh Stran* **1,** 84–96.

Naumov, B. N. (1980). Etapy Razvitiia Sistemy Malykh Elektronnykh Vychislitel'nykh Mashin. *Vychislitel'naia Tekhnika Sotsialisticheskikh Stran* (8), 5–10.

Naumov, B. N. (1984). Kolonka Redaktora. *Mikroprotsessornye Sredstva i Sistemy* (2), 2.

Naumov, B. N., Glukhov, Iu. N., Kabalevskii, A. N., and Panferov, B. I. (1979). Upravliaiushchie Kompleksy SM-3 i SM-4. *Vychislitel'naia Tekhnika Sotsialisticheskikh Stran* (6), 62–67.

Nesterov, P. V., *et al.* (1986). "Mikroprotsessory: Kn. 1. Arkhitektura i Proektirovanie mikro-EVM." Vysshaia Shkola, Moscow.

Nikitin, A. N., and Ostrovskii, V. P. (1988). Kompleks programmno-apparatnoi otkladki mikroprotsessornykh sistem na baze PEVM "Iskra-226". *Pribory i Sistemy Upravleniia* (4), 29–30.

Novak, S. (1983). Results and outlooks in Computer Technology. *Mechanizace a Automatizace Administrativy* (12), 453–454. (This article is a summary of an article published earlier in *Rechentechnik und Datenverarbeitung.*)

Operat. (1988a). Operatsionnaia Sistema "Al'fa-DOS." (brochure) VDNKh CCCP, (2).

Operat. (1988b). Operatsionnaia Sistema M86. (brochure) VDNKh SSSR, (2).

Oprishko, A. A., Afonin, L. A., Arekel'ian, V. V., Babaiants, A. B., and Pis'mennyi, V. V. (1987). Sostoianie i perspektivy avtomatizatsii biotekhnologicheskikh protsessov. *Pribory i Sistemy Upravleniia* (11), 40–41.

Ostapenko, G. P., and Filinov, E. N. (1987). Sovershenstvovanie Programmnogo Obespecheniia SM EVM. *Vychislitel'naia Tekhnika Sotsialisticheskikh Stran* **21**, 48–55.

Ostapenko, G. P., and Fridman, A. L. (1985). Instrumental'naia Sistema Postroeniia SUBD na Malykh i Puti Povysheniia ee Effektivnosti. *Vychislitel'naia Tekhnika Sotsialisticheskikh Stran* **17**, 90–95.

Ostrovskii, M. A. (1988). Mikroprogrammnaia struktura arifmetiko-logicheskogo protsessora VK SM 1700. *Pribory i Sistemy Upravleniia* (3), 11–12.

Paket. (1988a). Paket Prikladnykh Programm SLOG. (brochure) VDNKh SSSR(2).

Paket. (1988b). Paket Prikladnykh Programm ABAK. (brochure) VDNKh SSSR (2).

Pankratov, V. S., *et al.* (1987). Opyt Primeneniia Personal'nykh EVM. *Gazovaia Promyshlennost'* (5), 26–29.

Personnal'naia mikro-EVM Neiron I9.66. (brochure) Minpromsviazi.

Petrov, M. (1988). Nuzhen li spetsshkole "Agat." *Informatika i Obrazovanie* (1), 125.

Phister, M., Jr. (1979). "Data Processing Technology and Economics" (2nd ed.). Digital Press, Bedford, Massachusetts.

Pogorelyi, S. D., Slobodianiuk, A. I., Suvorov, A. E., and Iurasov, A. A. (1986). Personal'naia EVM "Neiron I9.66." *Mikroprotsessornye Sredstva i Sistemy* (4), 16–19.

Pogudin, Iu. M. (1987). MIASS—Sistemma Mikroprogrammirovanniia na Iazyke AMDASM *Mikroprotsessornye Sredstva i Sistemy* (5), 19–23.

Polosin, A. N., *et al.* (1986). Uchebnyi komp'iuter "elektronika UKNTs." *Mikroprotsessornye Sredstva i Sistemy* (6), 14–16.

Poom, K. E., Moor, A. E., Rebane, R. V., and Arulaane, T. E. (1986). Operatsionnaia Sistema dlia PEVM "Iskra 226." *Mikroprotsessornye Sredstva i Sistemy* (6), 21.

Popov, A. A., *et al.* (1984). Dialogovye Vychislitel'nye Kompleksy "Elektronika NTs-80-20." *Mikroprotsessornye Sredstva i Sistemy* (4), 61–64.

Popsuev, A. N. (1985). Opyt i Perspektivy Tsentralizovannogo Obespecheniia Razrabotchikov ASU Programmnymi Sredstvami. *Upravliaiushchie Sistemy i Mashiny* (4), 9–14.

Pravda. (1985). June 12, 2.

Presnukhin, D. L., ed. (1986a). "Mikroprotsessory: Kniga 1—Arkhitekura i Proektirovanie mikro-EVM." Vysshaia Shkola, Moscow.

Presnukhin, D. L., ed. (1986b). "Mikroprotsessory: Kniga 2—Sredstva Sopriazheniia, Kontroliruiushchie i Informatsionno-Upravliaiushchie Sistemy." Vysshaia Shkola, Moscow.

Presnukhin, L. N., and Shakhnov, V. A. (1986). "Konstruirovanie Elektronnykh Vychislitel'nykh Mashin i Sistem." Vysshaia Shkola, Moscow.

Presnukhin, L. N., *et al.* (1985). Laboratoriia po Izucheniiu Mikroprotsessornykh Komplektov s Fiksirovannym Naborom Komand. *Mikroprotsessornye Sredstva i Sistemy* (1), 77–81.

Priklad. (1985). "Prikladnoe Programmnoe Obespechenie Edinoi Sistemy EVM i Sistemy Mini-EVM." Vypusk 9, Koordinatsionnyi Tsentr Mezhpravitel'stvennoi Komissii po Sotrudnichestvu Sotsialisticheskikh Stran v Oblasti Vychislitel'noi Tekhniki, Moscow.

Prokhorov, N. L. (1987). Perspektivy Razvitiia SM EVM. *Vychislitel'naia Tekhnika Sotsialisticheskikh Stran* **21**, 3–12.

Prokhorov, N. L. (1988a). Osobennosti Arkhitektury i Programmnogo Obespecheniia Vychislitel'nogo Kompleksa. *Mikroprotsessornye Sredstva i Sistemy* (2), 6–9.

Prokhorov, N. L. (1988b). Funktsional'nye Vozmozhnosti i Sostav VK SM 1700. *Pribory i Sistemy Upravleniia* (3), 2–3.

Prokhorov, N. L., and Landau, I. Ia. (1984). MikroEVM SM-1800 i Ee Programmnoe Obespechenie. *Mikroprotsessornye Sredstva i Sistemy* (2), 28–30.

Prokhorov, N. L., and Smirnov, E. B. (1986). SM EVM: Sostoianie i Perspektivy Razvitiia. *Pribory i Sistemy Upravleniia* (2), 8–11.

Proleiko, V. M. (1984). Mikroprotsessornye Sredstva Vychislitel'noi Tekhniki i ikh Primenenie. *Mikroprotsessornye Sredstva i Sistemy* (1), 11–16.

Przhiialkovskii, V. V., ed. (1987). "Elektronnaia Vychislitel'naia Tekhnika" **1.** Radio i Sviaz', Moscow.

Pykhtin, V. Ia. (1986). ES 1840—Bazovaia personal'naia EVM edinoi sistemy. *Mikroprotsessornye Sredstva i Sistemy* (4), 15–16.

Raikov, D. D., and Rubanov, V. O. (1976). Prikladnye Programmy v Sisteme Programmnogo Obespecheniia ES EVM. *Upravliaiushchie Sistemy i Mashiny* (2), 33–39.

Rakovskii, M. E. (1979). Mezhdunarodnie Sotrudnichestvo Sotsialisticheskikh Stran v Oblasti Vychislitel'noi Tekhniki: Desiatiletie Sotrudnichestva. *Vychislitel'naia Tekhnika Sotsialisticheskikh Stran* **6,** 7–17.

Raud, R. K. (1982). The State of the Art in Microcomputer Programming. *Programming. and Computer Software* (7). (Translation of *Programmirovanie* (5), 31–43.)

Riabov, Ia. (1981). Problemy Uskoreniia Nauchno-tekhnicheskogo Progressa: Sotrudnichestvo Sotsialisticheskikh Stran v oblasti Vychislitel'noi Tekhniki—utogo i perspektivy. *Ekonomicheskoe Sotrudnichestvo Stran—Chlenov SEV* (2), 20–25.

Romanov, V. Iu., *et al.* (1985). Personal'naia EVM "Irisha": Periferiinye Ustroistva, Istochnik Pitaniia. *Mikroprotsessornye Sredstva i Sistemy* (3), 53–59.

Romanov, V. Iu., *et al.* (1986). Graficheskie Vozmozhnosti Personal'noi EVM "Irisha." *Mikroprotsessornye Sredstva i Sistemy* (1), 61–72.

Romashkin, F. Z., and Giglavyi, A. V. (1987). O Razvitii Programmnogo Obespecheniia dlia MikroEVM. *Pribory i Sistemy Upravleniia* (5), 2–4.

Rudins, G. (1970). Soviet Computers: A Historical Survey. *Soviet Cybernetics Review* (1), 6–44.

Rukavishnikov, V. D. (1988). Organizatsiia vychislitel'nogo protsessa v mnogomashinnom komplekce no baze EVM CM 1800. *Pribory i Sistemy Upravleniia* (4), 1–3.

Safonov, V. O., and Tsoi, V. N. (1987). Nekotorye Problemy Razvitiia Iazykov i Sistem Programmirovaniia. *Vychislitel'naia Tekhnika Sotsialisticheskikh Stran* **21,** 88–94.

Samarskii, A. A. (1984). Problemi primeneniia vychislitel'noi tekhniki. *Vestnik Akademii Nauk* (11), 17–29.

Sasov, A. Iu. (1986) Mikrotomografiia i Tsifrovaia obrabotka izobrazhenii na mikroEVM "Iskra 226." *Mikroprotsessornye Sredstva i Sistemy* (1), 53–58.

Savel'ev, A. Ia., ed. (1987a). "Osnovy informatiki." Kniga 2, Vysshaia Shkola, Moscow.

Savel'ev, A. Ia., ed. (1987b). "Elektronnye Vychislitel'nye Mashiny: Sredstva Obshcheniia s EVM." Book 6, Vysshaia Shkola, Moscow.

Savinov, V. I., and Gritsyk, V. V. (1988). Sistema vyvoda graficheskoi informatsii v VK na baze mikroEVM "Elektronika-60". *Pribory i Sistemy Upravleniia* (1), 33–34.

Selivanov, Iu. P. (1987). Novye tekhnicheskie sredstva ES i SM EVM. *In* "Elektronnaia Vychislitel'naia Tekhnika" **1** (V. V. Przhiialkovskii, ed.), pp. 210–216. Radio i Sviaz', Moscow.

Semenkov, O. I., *et al.* (1987). Mikroprotsessornaia Graficheskaia Stantsiia GT-80. *Mikroprotsessornye Sredstva i Sistemy* (4), 54–56.

Shakhnov, V. A. (1984). Razvitie i Primenenie Mikroprotsessorov i Mikroprotsessornykh Komplektov BIS. *Mikroprotsessornye Sredstva i Sistemy* (1), 17–22.

Shekhovtsev, K. (1988). O pechal'nom i zabavnom. *Informatika i Obrazovanie* (4), 126–127.

Shirokov, F. (1988). Why be Tricky? *NTR* (8), 19 April–2 May, 4. (As translated in JPRS-UST-88-011, 42–44).

Shkamarda, A. N. (1986). Shestnadtsatirazriadnye MikroEVM Semeistva SM 1800. *Mikroprotsessornye Sredstva i Sistemy* (5), 6–10.

Signaevskii, V. A. (1988). Ob upravlenii pamiat'iu v EVM SM 1700. *Pribory i Sistemy Upravleniia* (3), 5–6.

Sinitsin, N. V., Petropavlovskii, V. P., and Nikitin, A. M. (1987). *Novoe v Zhizni, Nauke, Tekhnike: Seriia Radioelektronika i Sviaz* (1), as translated in JPRS-UCC-87-017, September 23, 1987, 32–80.

"Sistemnye Programmnye Sredstva ES EVM i SM EVM." (1987). (1), Mezhdunarodnoe tsentr nauchnoe i tekhnicheskoi informatsii, Moscow.

Smirnitskii, E. K. (1986). "Dvenadtsataia Piatiletka." Politisdat, Moscow.
Sobranie Postanovlenii Pravitel'stv SSSR Otdel Pervyi (1987). **29,** 596–603.
Solomatin, N. M., *et al.* (1987). "Vybor MikroEVM dlia Informatsionnykh Sistem." Vysshaia Shkola, Moscow.
Solov'ev, G. N., ed. (1985). "Skhemotekhnika EVM." Vysshaia Shkola, Moscow.
Sovetskaia Belorussiia (1983). 3 April, 1.
Sovetskaia Belorussiia (1984). 8 January, 1.
Sovetskaia Litva (1985). 22 February, 4.
Stapleton, R. A. (1985). Soviet and East European Microcomputer Systems. *Signal* (4), 69–76.
SUBD. (1988). SUBD REBUS. *Pribory i Sistemy Upravleniia* (7), advertising insert.
Sulim, M., Lazarev, A., and Safanov, V. (1986). Znakom'tes: "Korvet." *Informatika i Obrazovanie* (1), 74–75.
Szamitastechnika (1984). November, 1, 10.
Szuprowicz, B. O. (1973). Soviet Bloc's RIAD Computer System. *Datamation.* September, 80–85.
Talov, I. L., *et al.* (1982). Novaia Vysokoproizvoditel'naia miniEVM "Elektronika-79" i Effectivnost' ee Primeneniia v Sistemakh Skhemotekhnicheskogo Proektirovaniia BIS. *Upravliaushchie Sistemy i Mashiny* (6), 105–108.
TASS Report. (1983). July 7.
Tilinin, D. A. (1986). Personal'naia EVM "Okean 240." *Mikroprotsessornye Sredstva i Sistemy* (2), 24–25.
Tilinin, D. A., *et al.* (1986). Personal'naia EVM "Okean 240." *Mikroprotsessornye Sredstva i Sistemy* (4), 69–78.
Tilinin, D. A., *et al.* (1987). PEVM "Okean 24": Konstruktsiia i Metodika Otladki. *Mikroprotsessornye Sredstva i Sistemy* (3), 77–86.
Tolstov, V. (1987). Elektronika u Nas Doma. *Izvestiia,* 21 March, 4.
Tolstykh, B. L., *et al.* (1987). "Mini- i Mikro EVM Semeistva 'Elektronika.'" Radio i Sviaz', Moscow.
USiM. (1976). K 25-letiiu Sozdaniia Pervoi Otechestvennoi EVM. *Upravliaiushchie Sistemy i Mashiny* (6), 3–6.
USiM. (1977). Ordena Lenina Institutu Kibernetiki AN Ukrainskoi SSR—Dvadtsat' Let. *Upravliaiushchie Sistemy i Mashiny* (6), 3–6.
USiM. (1982). Ordena Lenina Institutu Kibernetiki im. V. M. Glushkova—25 Let. *Upravliaiushchie Sistemy i Mashiny* (5), 3–8.
Veitsman, V. (1988). Start v informatiky. *Informatika i Obrazovanie* (5), 65–71.
Velikhov, E. P., *et al.* (1986). Personal'nyi komp'iuter v sisteme avtomatizatsii fizicheskogo eksperimenta. *Mikroprotsessornye Sredstva i Sistemy* (1), 34–36.
Velikhov, E. P. (1987a). Ob osnovykh itogakh razvitiia fiziko-tekhnicheskikh i matematicheskikh nauk v 1986 g. *Vestnik Akademii Nauk, SSSR* (7), 19–31.
Velikhov, E. P. (1987b). O Zadachakh Akademii Nauk SSSR v Svete Reshenii Iiun'skogo (1987) Plenuma TsK KPSS. *Vestnik Akademii Nauk, SSSR* (12), 14–26.
Velikhov, E. P. (1988). O dostizheniiakh Akademii Nauk SSSR, v oblasti fiziko-tekhnicheskikh i matematicheskikh nauk v 1987. *Vestnik Akademii Nauk SSSR* (7), 22–26.
Verner, V. D., *et al.* (1986). "Mikroprotsessory." Vysshaia Shkola, Moscow.
Vestnik Statistiki. (1988). Nauchno-tekhnicheskii progress. (7), 58–69.
Vigdorchik, G. V., *et al.* (1987a). Personal'naia EVM PK-11. *Mikroprotsessornye Sredstva i Sistemy* (1), 16–18.
Vigdorchik, G. V., *et al.* (1987b). Personal'naia EVM "Kvant." *Mikroprotsessornye Sredstva i Sistemy* (1), 18–20.
Vinokurov, V., and Zuev, K. (1985). Aktual'nye problemy razvitiia vychislitel'noi tekhniki. *Kommunist* (5), 18–29.
Vodiankin, A. G., and Moiseenko, V. I. (1987). Uchebnaia Lokal'naia Set' MikroEVM. *Mikroprotsessornye Sredstva i Sistemy* (4), 83–84.

Vorob'ev, A. D., *et al.* (1987). PPEVM "Istra": Arkhitektura, Tekhnicheskie Kharakteristiki. *Mikroprotsessornye Sredstva i Sistemy* (1), 15–16.

Vyshnevskii, Iu. L. (1985). Printsipy sistemy MARS v arkhitekture vysokoproizboditel'nogo protsessora. *In* "Kibernetika i vychislitel'naia tekhnika" (A. Mel'nikov, ed.) (1), pp. 79–87. Nauka, Moscow.

Wasowski, S., ed. (1970). "East-West Trade and the Technology Gap." Praeger Publishers, New York.

Wolcott, P., and Goodman, S. E. (1988). High-speed Computers of the Soviet Union. *Computer*, September, 32–41.

Yasmann, V. (1985). Soviet Leaders Grapple with the Scientific and Technological Revolution. Radio Liberty Research Report 192/85, June 14.

Yasmann, V. (1987). Personal Computers in the USSR—Will Help Come From the West? Radio Liberty Research Report 388/87, September 23.

Zamorin, A. P., Selivanov, Iu. P., and Lokshin, Ia. P. (1984). Novye vychislitel'nye mashiny, razrabotannye v SSSR. *Vychislitel'naia Tekhnika Sotsialisticheskikh Stran* **16,** 159–166.

Zavartseva, N. M., and Ivanova, S. V. (1986). Vsesoiuznaia Nauchno-tekhnicheskaia Konferentsiia "Problemy Sozdaniia i ispol'sovaniia mini- i mikro-EVM." *Pribory i Sistemy Upravleniia* (7), 42–44.

Zonis, V. S. (1988). Protsessor arifmetiki s plavaiushchei zapiatoi VK SM 1700. *Pribory i Sistemy Upravleniia* (3), 4–5.

AUTHOR INDEX

Numbers in italics indicate the pages on which the complete references are given.

C

D

E

F

G

L

M

N

O

P

R

S

T

U

V

W

Y

Z

This Page Intentionally Left Blank

SUBJECT INDEX

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Contents of Previous Volumes

Volume 1

Volume 2

Volume 3

Volume 4

Volume 5

Volume 6

Volume 7

Volume 8

Volume 9

Volume 10

Volume 11

Volume 12

Volume 13

Volume 14

Volume 15

Volume 16

Volume 17

Volume 22

Volume 23

Volume 24

Volume 25

Volume 26

Volume 27

Volume 28

This Page Intentionally Left Blank