Final Report: Development of a Practical Computer Software Verification System

TR-90-5429-11

Dan Craigen

Release date: March 1990

ORA Canada
267 Richmond Road, Suite 100
Ottawa, Ontario K1Z 6X3
CANADA

DSS Contract No. W2207-7-AF78/01-SV

Abstract

The purpose of this report is to present an overview of contract W2207-7-AF78/01-SV, entitled "Development of a Practical Computer Software Verification System."

1 Introduction

The purpose of this report[1] is to present an overview of contract W2207-7-AF78/01-SV, entitled "Development of a Practical Computer Software Verification System." This contract was funded by the Canadian Department of National Defence (DND) with Vincent Taylor acting as the Technical Authority. We will call this contract the "EVES contract" for the remainder of this report.[2]

The contract started in November 1987 and was completed in the fall of 1989. It was initially awarded to the Trusted Systems Group of I. P. Sharp Associates Limited (IPSA), but was transferred to Odyssey Research Associates (ORA) as of May 1989. The transfer of the contract was a consequence of the change of ownership of IPSA (to Reuters, plc.) in the summer of 1987. Reuters ownership of IPSA resulted in a change of business emphasis at IPSA and, consequently, it was necessary for the individuals involved in this contract to find a new affiliation. In May 1989, the individuals involved in the EVES contract moved from IPSA to ORA. The initial project manager, David Bonyun, left the project in August 1988 and project management was taken over by Dan Craigen. The members of the technical team remained constant throughout the contract and were Dan Craigen, Sentot Kromodimoeljo, Irwin Meisels, Bill Pase, and Mark Saaltink. Karen Summerskill provided support as technical editor and librarian. Denine Wrixon provided administrative support.

The EVES contract was a successor to contract W2207-6-AF08/01-SV, entitled "Software Tools for the m-Verdi Computer Language," which was completed in late 1987. The "software tools" contract led to the development of a prototype verification system, called m-EVES, which demonstrated the feasibility of developing a system which brought together ideas from automated theorem proving, language design, and formal semantics. m-EVES has been a successful prototype and pedagogical system, and has been distributed to sites in Australia, Canada, the Federal Republic of Germany, the United Kingdom, and the United States. Two papers [CKM* 87, PS 89] present surveys of the work performed.

The purpose of the EVES contract was to build upon our experiences with m-EVES and to develop a verification system that had greater emphasis towards production concerns. For example, the design of the programming and specification language m-Verdi [Cra 87] was simplified in some instances because of the "prototype" and "proof of concept" aspects of m-EVES. One of the main goals of the EVES contract was to design a language, called Verdi, which built upon the m-Verdi experiences and improved expressibility. The second major goal was to modify and enhance the automated theorem proving support. The resulting prover is called NEVER.

The remainder of this report is divided as follows:

`Deliverables:` identifies the contract deliverables. The list of deliverables includes references to the technical documentation produced. As a consequence, the technical details described in those reports are only summarized here.

`Design of Verdi:` describes some of the background considerations that motivated various Verdi language design decisions, presents a brief overview of Verdi, and compares and contrasts Verdi with m-Verdi.

`Development of NEVER:` summarizes various experiments that were performed and the changes in emphasis from m-NEVER to NEVER.

`Development of EVES:` summarizes the work involving the implementation of the non-theorem proving aspects of the verification system.

[1] This report describes research performed by Odyssey Research Associates for the Canadian Department of National Defence under contract W2207-7-AF78/01-SV.

[2] "EVES" is an acronym for "Environment for Verifying and Evaluating Software."

`Background research:` summarizes some of the background work that was performed during this contract (primarily, the investigation of various technical alternatives).

`Academic Participation and Examples:` summarizes the participation of the EVES group in various conferences, demonstrations of the verification system, and various examples that were tried during the period of the contract.

`Conclusions:` includes various concluding remarks pertaining to the project.

2 Deliverables

The following items were delivered as a part of the EVES contract:

- Monthly progress reports.
- Copies of computer listings for the theorem prover (NEVER) and, in general, for EVES.
- *Software Manual for EVES:* ORA Technical Report TR-89-5429-12, October 1989.
- *A Formal Description of Verdi:* ORA Technical Report TR-89-5429-10, October 1989.
- *Reference Manual for the Language Verdi:* ORA Technical Report TR-90-5429-09, February 1990
- This final report. *Final Report: Development of a Practical Computer Software Verification System:* ORA Technical Report TR-90-5429-11, March 1990.
- A tutorial on EVES (which was held in November 1989).

It should be noted that the user manual for NEVER was incorporated into the reference manual for the language Verdi. The user manual was incorporated because a Verdi language design decision resulted in the interface language for EVES being Verdi. (This differs from the m-EVES decision, and is a result of m-EVES experiences, where m-Verdi and the m-EVES command language were different in syntax and scope.)

3 Design of Verdi

This section presents the motivation for our choice of set theory as the basis for Verdi and for our choice of a Lisp-like syntax for Verdi. In addition, a brief summary of the language and a comparison of m-Verdi and Verdi are presented.

3.1 Choice of Set Theory over HOL

One of our main goals in designing Verdi was to improve expressibility (in comparison with m-Verdi) in both the specification and programming components of the language.

From the specification perspective, we envisioned two options for achieving the above:

- the use of a higher-order logic (specifically, type theory), or
- the use of axiomatic set theory.

Our initial feelings were to proceed with a "type theory" approach. However, further considerations led to the choice of an untyped axiomatic set theory framework. Some of these considerations were

- set theory has been widely used as a foundation for mathematics;
- set theory is well understood;
- many existing mathematics and computing science books use set theory to describe concepts;
- more is known about automated proofs within a first-order framework than in higher-order systems;
- there are algorithms for determining unification within a first-order framework, but unification is undecidable in type theory.
- there are limited existing applications of type theory; and
- there are still unresolved theoretical issues relating to type theory.

Consequently, we chose a ZFC axiomatization (i.e., a Zermelo-Fraenkel axiomatization with the "Axiom of Choice") and Verdi is based on a first-order logical framework.

From the programming perspective, the decisions were much simpler. There were some obvious areas of improvement which could be handled at (reasonably) low technical risk. These areas included "for loops," improved enumeration types, complete support of subvariables, mutually recursive executable procedures and a library facility.

The choice between higher-order logic (in this instance, type theory) and a version of untyped set theory was the single most important decision to be made during the project. As noted above, our initial leanings had been to proceed with the higher-order logic approach and, in fact, the work on the theorem prover NEVER (discussed in Section 4) started from this assumption. Much of the language and logic work performed by Saaltink (primarily) and Craigen focused on the advantages and disadvantages of proceeding with either framework. As the choice was unclear, it took approximately a year before, with a degree of apprehension, we selected the untyped set theory approach. It is, however, important for the reader to understand that the work on incorporating higher-order capabilities into the theorem prover NEVER did *not* go for naught. Many of the prover's algorithms work within a higher-order framework (though the general capabilities are not available from EVES) and, as a consequence, have resulted in code which is simpler and faster than that occurring in m-NEVER. Besides resulting in better quality code, our decision has been hedged. To use Kromodimoeljo and Pase's phraseology: "NEVER is higher-order ready."

3.2 Verdi Syntax

We decided to proceed with an "abstract syntax" approach to Verdi. The abstract syntax is based on Lisp with the intention that we can use some of the tools that have been designed for that language. The EVES R&D effort was not to include the development of a large collection of language specific tools.

Further, by defining an abstract syntax, we are opening the possibility for organizations that wish to enhance EVES to develop their own concrete syntax, to translate their concrete syntax into our abstract syntax, and to use their proprietary tools. For example, an organization that has a bit mapped graphics screen might support a wide spectrum of mathematical notations. It is also possible that the definition of the abstract syntax will help to identify the boundary between those aspects of the verification system that can be modified by such organizations and those aspects that must remain inviolate (for soundness reasons).

Moreover, the abstract syntax approach allows for greater commonality of notation between the mathematics document [Saa 89], reference manual [Cra 90a], and the representation used within the verification system.

Interestingly, early experiences with Verdi, have shown a marked preference for the Verdi notation when compared with that of m-Verdi.[3]

3.3 Verdi Overview

This section of the report is based on a similar section in the Verdi reference manual [Cra 90a]. The reader is directed to the reference manual (specifically, Chapter 1) for a discussion of the conceptual framework for Verdi.

Verdi declarations result in the modification of a theory[4] by extending the vocabulary and adding further axioms. The proof obligations associated with each declaration mandate that each extension to a theory is a (semantic) conservative extension (see [Saa 89]).

The various function declarations introduce functions. Functions that are used in an executable context must have been introduced by a "typed function declaration." Such a declaration associates type information with the parameters and the result of the function. The result of an executable function application is predictable only if the application is legal. By legal, it is meant that the parameters satisfy constraints (determined by a function "pre-condition"). Non-executable functions may be defined recursively and, by using function recursion groups, through mutual recursion. Certain functions, defined as part of the initial theory, are called "wide domain functions" and are those functions that have more than one signature; these functions are similar to "operators" in classical programming languages.

A procedure declaration introduces a procedure and is always executable. Procedures may be defined recursively and, by using procedure recursion groups, through mutual recursion. Procedures may have "open-array" parameters.

An axiom declaration restricts the possible structures for the names in a vocabulary.

While Verdi is, in general, an untyped language, typing information is necessary for syntactic forms that are to be executed. A type name denotes a set of values.

The Bool, Int and Char types belong to the initial theory. The Bool type denotes the logical truth values. The Int type denotes the set of mathematical integers. The Char type denotes the ASCII character set. Other types are introduced through an enumeration declaration, an array declaration, or a record declaration. Verdi type sameness is a variant of structural sameness (since the structures of records and enumerations are not used in sameness checking). Hence, Verdi type sameness is a compromise between the extremes of name sameness and structural sameness.

An expression is a Verdi sentence that can be evaluated (using a structure and a state) to produce a value. The expressions are character literals, numerals, strings, identifiers (denoting variables), function applications, and quantifications. Certain expressions are called *manifest expressions* and are mandatory in certain instances where expressions are syntactically required (i.e., case labels and array bounds). Manifest expressions can be evaluated at compile time.

A statement is a Verdi sentence that denotes one or more execution steps and determines, in part, the ordering of the execution steps. It is through the execution of statements (using a structure and a state) that the values associated with the observables and states are modified. The Verdi statements are exit (from a loop), return (from a procedure), abort (the program), note (a form of annotation), assignment, procedure call, block, conditional, case, loop, and two forms of "for" loop.

A library is a repository for library objects. Library objects are collections of Verdi declarations and, through the use of the **load** library command, may depend upon other library objects. The **load** library command introduces the declarations of the associated library specification object (and any necessary subsidiary library specification objects) to a current theory. Library objects are one form of support for modularization and abstraction.

[3] It is only fair, however, to add the caveat that the early experiences have been with individuals who are expert (or, at least, familiar) with LISP.

[4] A theory is a collection of symbols (the vocabulary), and axioms pertaining to those symbols.

Verdi also supports the various system, prover, and library commands that are used to interact with EVES. Verdi is the interface language used with EVES.

3.4 Verdi improvements over m-Verdi

Summarizing the improvements (and differences) resulting from the Verdi design, we have:

- The logical basis is untyped first-order logic, with Zermelo-Fraenkel (and the "Axiom of Choice" built-in). Hence, the specification component of the language is more expressive.
- Typing is not completely absent from the language, as executable constructs are still subject to type checks. The typing system is more flexible than m-Verdi's (a mix of name-sameness and structure-sameness) and includes a provision for "open-array" parameters.
- A library mechanism has been added. This addition goes much further than m-Verdi's package mechanism as it supports the separate development of library units.
- Procedures and non-executable functions can be recursive or mutually recursive.
- Domain errors during execution are handled. An implementation is allowed to abort during a calculation when a function is applied to arguments outside of its domain. Various Verdi proof obligations are generated to show that such mis-applications will not occur. This approach has allowed us to eliminate the separate executable, non-executable approach (e.g., `plus` and `eplus`) of functions found in m-Verdi.
- "For loops" have been added and the definition of enumeration types adjusted to admit more reasonable proofs of procedures iterating over the full range of an enumeration type.
- Subvariables are fully supported. In m-Verdi, values could be assigned to subvariables, but subvariables could not be passed to "pvar parameters." Verdi is more liberal.
- The m-Verdi form of variable declaration has been removed.
- The loop proof rule has been strengthened, so that less information needs to appear explicitly in the invariant. Further, it is now possible to prove nested loops; this could not be done in m-Verdi.

It is important to note that, while we have substantially improved the expressibility of the language, we have maintained the sound mathematical basis [Saa 89]. In the mathematics document [Saa 89], a denotational description of the language is presented, as is a proof theory. The proof theory has been shown to be sound relative to the denotational model. As those familiar with the m-Verdi mathematics [Saa 87] will observe, the mathematical framework for Verdi is more complex (as a result of the additional capabilities) than that for m-Verdi.

Verdi was designed by Dan Craigen and Mark Saaltink, with material assistance from Irwin Meisels.

4 Development of NEVER

As reported in the previous section, the logical framework for EVES is quite different from m-EVES. Perhaps the single most important distinction is the move away from a strongly typed framework to one which is untyped. This distinction is also prevalent in comparing NEVER with m-NEVER.

Once it was clear which logical framework was to be used for EVES, significant research and development were focused on how one should proceed with automated deduction within a typeless

framework—especially since some of the most useful capabilities of m-NEVER, the various decision procedures, are strongly dependent on knowing the type of an expression.

Consequently, the typing framework, which was so integral to m-NEVER, was removed. An analysis of approaches used by the Boyer-Moore prover, which is also untyped, was undertaken and, based on that analysis, a "type prescription mechanism" incorporated into NEVER. The basic idea was that NEVER (and Verdi) would support type predicates and the type prescription mechanism could be used to determine the type, if one exists, of an expression. The determination of typing information for an expression allows for the application of the type specific decision procedures mentioned above.[5]

In addition to the major restructuring of the prover, a new initial theory needed development and the prover needed to handle the changes to theories resulting from the various Verdi declarations. The initial theory, besides consisting of the vocabulary and axioms that are built into the language, has an extensive heuristic component. Numerous experiments were performed to determine how the heuristics should be included. Many of the decisions resulted from earlier experiments describing set theory within m-EVES (and using m-NEVER). (The initial theory that was incorporated in EVES (Version 1.0) is described in the Verdi reference manual [Cra 90a].)

As it was initially expected that Verdi would be higher-order, early work on NEVER focused on the addition of a higher-order capability. Even though higher-order is not a part of the Verdi language design, it is still a part of the implementation of the prover. The reasons justifying the inclusion are the improvement in the code and the reduction of special casing.

Various other modifications and experiments were conducted, including the following:

- The conversion of some small examples from m-EVES to EVES (including the "flow modulator" and "arraymin" examples, described in the Verdi reference manual [Cra 90a], and the sequence theory).
- The implementation of nested histories (needed for libraries).
- The implementation of a different way of handling Boolean equalities. Previously, the prover converted Boolean equalities into "if" forms, thereby causing numerous case splits. The new method for handling Boolean equalities, which uses the "deductive database," does not cause case splits and is faster.
- The implementation of short-circuiting of subgoals[6] and the caching of forward rules. For the system's handling of a quicksort specification, the running time of the system was improved by 50%.
- The modification of the equality substitute command so that the substitutions used by NEVER are reported to the user.

NEVER was implemented by Sentot Kromodimoeljo and Bill Pase.

5 Development of EVES

In addition to the language and theorem prover work reported above, there were other modifications and improvements that differentiate EVES from m-EVES. These include:

- The implementation of a faster EVES parser and prettyprinter.
- The implementation of a Zmacs interface to EVES (with improvements over that of m-EVES).

[5] Some parallel experiments involved the implementation of a Boyer-Moore mode on top of the verification system [Fis 89]. Feedback relating to performing proofs in a typeless framework was obtained from these experiments.

[6] For example, if a conjunct in a subgoal is shown to be `false`, there is no need to prove the remaining conjuncts.

- The implementation of Lempel/Ziv compression to compress the size of freeze files. The implementation of this compression technique increased the speed of the freeze and thaw commands. (The requirement to implement a more efficient storage representation was a result of our LOCK experiments. The LOCK specifications were sufficiently large that the previous storage scheme was deemed inadequate.)

- The implementation of a library mechanism. However, it must be admitted that the implementation is not without fault. The current problems mainly concern the multiple libraries and distributing libraries. Multiple libraires can be fooled; for example, a unit (called it U1) from library L1, used by a unit U2 from library L2, may be changed without telling L2. Distributing libraries are a problem because of the lack of logical path names. Further study (especially into logical path names) of the library mechanism is necessary. Conventional use of the library mechanism is sound.

- The implementation of a mechanism that catches keyboard interrupts and, hence, permits the interruption of abortive proof efforts.

The EVES development was implemented primarily by Sentot Kromodimoeljo, Irwin Meisels, and Bill Pase.

6 Background Research

This section summarizes some of the background investigations that were pursued through the project, though not necessarily included in EVES. The project status reports present further details on most of the matters described in this section.

- As part of a separate contract, funded by the United States Navy, we investigated the languages CSP and CCS to see how they handled concurrency. We decided that, at this juncture, it was premature to consider the addition of a concurrency feature to Verdi. There were two specific reasons for this decision. First, there is still disagreement over what the suitable semantic models for CSP and CCS should be and, as a consequence, even a simple question pertaining to equivalence of processes has not been satisfactorily resolved [BR 83, Pra 86, BIM 88]. Second, until very recently, there had not been any research on how to use abstraction on communication channels [AH 89].

- We investigated various semantic techniques used for nondeterminism, parallelism, and the role of powerdomains [Smy 78, AP 81].

- We investigated various library mechanisms. For example, we considered including parameterized packages. Unfortunately, we found this kind of expressiveness came at a high cost in complexity. The Verdi library mechanism supports the required functionality (of abstraction and modularity) and is not particularly complex. (The reasoning leading to our choice of library mechanism was discussed, in detail, during the November 1988 review meeting.)

- We investigated various higher-order and polymorphic logics. As noted above, we decided to forego this direction.

- We pursued various experiments with the Knuth-Bendix algorithm (especially with respect to Set Theory and Group Theory). These studies were to investigate the potential roles of the Knuth-Bendix algorithm on the kinds of rewrite rules used in EVES. It was determined that further investigation was warranted.

- We studied, prototyped, and experimented with efficient algorithms for unification. None of the algorithms studied were incorporated into EVES. For example, the experiments showed that, in practice, the 'efficient' algorithms were slower than the naive unification algorithm used by EVES. Specifically, three algorithms were investigated:

  1. The first algorithm prototyped was that of Martelli and Montanari. This was done as a comparison to our version of unification as done within the "e-graph".
  2. The second was an associative and commutative unification algorithm described by Stickel. This also involved Huet's work on generating basis solutions for a set of linear homogeneous Diophantine equations.
  3. The third was an algorithm developed by G. Escalada-Imaz and M. Ghallab. The algorithm has a near-linear worst case complexity. Our implementation of the algorithm is faster than our implementation of the Martelli-Montanari algorithm on all examples that we tested.

- We investigated different formulations of set theories (especially those with proper classes). Approaches using classes (e.g., [Fra 73]) were dropped as the modest gain in expressibility came at a continuing cost of proving that set objects were, in fact, sets.

- We investigated various non-standard logics, including temporal logics (specifically, the temporal logic called S_5). Temporal logic is not a part of EVES. There was an attempt to graft temporal logic on top of the first-order framework but this led to serious problems in representing temporal formulae on output (and after being manipulated by NEVER). Since significant modifications to the prover would be required to add a temporal logic feature, we decided not to proceed with these modifications.

7 Academic Participation and Examples

When involved in a research and development project such as the EVES project, it is important to maintain contact with others in the field and to participate in various academic activities. Members of the EVES group attended various conferences during the contract; published a number of papers (including [CKM* 87, Cra 88, PK 88, PS 89]); demonstrated EVES at both the Logic in Computer Science Conference (held at Cornell University) and the 9th International Conference on Automated Deduction (held at Argonne National Labs); demonstrated EVES to visitors from Australia, Canada, the United Kingdom, and the United States; held courses on the m-EVES system (drawing participants from both the United States and Canada); and were on program committees for various conferences and workshops. The EVES research group is held in high regard by others in the formal methods and automated deduction fields.

As the system evolved through the period of the contract, it was applied to various examples. As it was relatively late in the project when the Verdi language was integrated fully into EVES, many of the examples—while using new theorem proving and environment features—still were written in m-Verdi. As of this report, Verdi has been used for a "flow modulator," an "arraymin" program, sequence theory, and an interpreter [PK 90]. The interpreter is the largest and most complex example yet analyzed by EVES. While lines of text are a questionable metric, it is of interest to note that this effort required approximately 10,000 lines of code, specification, and system commands.

Other examples were tried, however. Probably of most interest were our experiments with the LOCK specification—a large non-interference style specification and proof. We wrote a Gypsy to m-Verdi translator, developed m-Verdi theories for the Gypsy variants of sequences, sets and mappings, and tried some of the proofs. We found that the interaction needed with the evolving EVES system was about six times better than that required for Gypsy. The example was also useful for indicating some weakness in our system and led to improvements in, for example, the form of freeze files.

Many other examples such as (various versions of) Quicksort, a ripple-carry adder (built from CMOS transistors), sequence theory and set theory were also successfully completed.

8 Conclusions

For most of this contract, the general working conditions within which the EVES research group found itself were unsettled. This was primarily a result of the change of ownership at I. P. Sharp Associates and the subsequent search for a new corporate parent. In this author's view, that all of the members of the technical team, along with Summerskill and Wrixon, remained with the project through to its completion, showed a high level of professionalism and commitment to the needs of the sponsoring department. I would like to take this opportunity to commend them for their performance.

Even with the aforementioned disruptions, the EVES effort was completed. The two major goals of developing the new language Verdi (with supporting mathematical descriptions) and developing the new prover NEVER were achieved.

There are some definite improvements when comparing EVES with m-EVES. Of particular note are the increase of expressibility embodied in Verdi and the supporting automated theorem proving support. What is missing, at this point, is a period of experimental application of EVES. Other reports to the Canadian government have suggested various possible research directions and the role of EVES in the longer term.[7]

References

[AH 89] L. Aceto and M. Hennessy. "Towards Action-Refinement in Process Algebras." In *Proc. 4th Annual Symposium on Logic in Computer Science*, IEEE Computer Society Press, 1989.

[AP 81] K. Apt and G. Plotkin. "A Cook's Tour of Countable Non-determinism." In *Proc. 8th ICALP*, Springer-Verlag, LNCS 115, 1981.

[BR 83] S. Brooks and W. Rounds. "Behavioral Equivalence Relations Induced by Programming Logics." In *Proc. 10th ICALP*, Springer-Verlag, LNCS 154, 1983.

[BIM 88] B. Bloom, S. Istrail, A. Meyer. "Bisimulation Can't Be Traced: Preliminary Report." In *Proceedings of the 15th Annual ACM Symposium on Principles of Programming Languages*, January 1988.

[CKM* 87] Dan Craigen, Sentot Kromodimoeljo, Irwin Meisels, Andy Neilson, Bill Pase, and Mark Saaltink. "m-EVES: A Tool for Verifying Software." In *Proc. 10th International Conference on Software Engineering*, Singapore, April 1988.

[Cra 87] Dan Craigen. *A Description of m-Verdi.* IPSA Technical Report TR-87-5420-02, November 1987.

[Cra 88] Dan Craigen. "An Application of the m-EVES Verification System." In *Proc. 2nd Testing and Verification Workshop*, Banff, Alberta, July 1988.

[Cra 90a] Dan Craigen. *Reference Manual for the Language Verdi.* ORA Technical Report TR-90-5429-09, February 1990.

[7] The reader is also referred to the report from the Halifax FM89 workshop [Cra 90b], held in July 1989, for discussions pertaining to the role of formal methods in the development of critical systems and for recommendations pertaining to the future development of formal methods technology.

[Cra 90b] Dan Craigen (Editor), Karen Summerskill (Assistant editor). *Report on The 1989 Workshop on Formal Methods and Critical Systems: FM89.* In preparation.

[Fis 89] Jason Fischl. *A Boyer-Moore Mode in m-EVES.* ORA Technical Report TR-89-5429-07, August 1989.

[Fra 73] A. A. Fraenkel, Y. Bar-Hillel and A. Levy. *Foundations of Set Theory.* North-Holland.

[PK 88] Bill Pase and Sentot Kromodimoeljo. "m-NEVER System Summary." In *Proc. 9th Conference on Automated Deduction*, Argonne, Illinois, May 1988. LNCS 310.

[PK 89] Bill Pase and Sentot Kromodimoeljo. *Software Manual for EVES.* ORA Technical Report TR-89-5429-12, October 1989.

[PK 90] Bill Pase and Sentot Kromodimoeljo. *Using the EVES Library Facility: A PICO Interpreter*, ORA Technical Report TR-90-5444-02, February 1990.

[PS 89] Bill Pase and Mark Saaltink. "Formal Verification in m-EVES." In *Current Trends in Hardware Verification and Automated Theorem Proving*, Graham Birtwistle and P.A. Subrahmanyan (eds.), Springer-Verlag, New York, 1989, pp. 268-302.

[Pra 86] Pratt. "Modelling Concurrency with Partial Orders." In *International Journal of Parallel Programming*, Volume 15, pp. 38-71, 1986.

[Saa 87] Mark Saaltink. *The Mathematics of m-Verdi.* IPSA Technical Report TR-89-5420-03, November 1987.

[Saa 89] Mark Saaltink. *A Formal Description of Verdi.* ORA Technical Report TR-89-5429-10, October 1989.

[Smy 78] M. Smyth. "Power Domains." *Journal of Computer and System Sciences*, Volume 16, pp. 23-36, 1978.