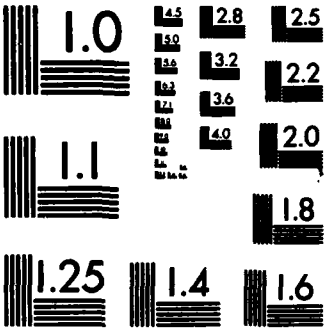
AD-A184 338 SDVS (STATE DELTA VERIFICATION SYSTEM) PROGRAM VERIFICATION SYSTEM DEVEL (U) AEROSPACE CORP EL SEGUNDO CA LAB OPERATIONS L G MARCUS 01 JUL 87 1/1

UNCLASSIFIED TR-0086(6778)-1 SD-TR-87-35 F/G 12/5 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

REPORT SD-TR-87-35

(12)

SDVS: Program Verification System Development FY '86 Final Report

L. MARCUS
Computer Science Laboratory
Laboratory Operations
The Aerospace Corporation
El Segundo, CA 90245

1 July 1987

SPACE DIVISION
AIR FORCE SYSTEMS COMMAND
Los Angeles Air Force Station
P.O. Box 92960, Worldway Postal Center
Los Angeles, CA 90009-2960

This report was submitted by The Aerospace Corporation, El Segundo, CA 90245, under Contract No. F04701-85-C-0086 with the Space Division, P.O. Box 92960, Worldway Postal Center, Los Angeles, CA 90009-2960. It was reviewed and approved for The Aerospace Corporation by A. J. Schiewe, Acting Director, Computer Science Labortory.

Lt David Rosenberg/CLVA was the project officer.

This report has been reviewed by the Public Affairs Office (PAS) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nationals.

This technical report has been reviewed and is approved for publication. Publication of this report does not constitute Air Force approval of the report's findings or conclusions. It is published only for the exchange and stimulation of ideas.

DAVID ROSENBERG, Lt, USAF
Project Officer
SD/CLVA

JOSEPH HESS, GM-15
Director, AFSTC West Coast Office
AFSTC/WCO OL-AB

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) TR-0086(6778)-1	5. MONITORING ORGANIZATION REPORT NUMBER(S) SD-TR-87-35
6a. NAME OF PERFORMING ORGANIZATION The Aerospace Corporation Laboratory Operations 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION National Security Agency
6c. ADDRESS (City, State and ZIP Code) El Segundo, CA 90245	7b. ADDRESS (City, State and ZIP Code) Ft. Meade, MD 20755
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Space Division 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F04701-85-C-0086
8c. ADDRESS (City, State and ZIP Code) Los Angeles, CA 90009-2960	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification) SDVS: Program Verification System Development-FY '86 Final Report	
12. PERSONAL AUTHOR(S) Marcus, Leo G.	
13a. TYPE OF REPORT 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) 1 July 1987 15. PAGE COUNT 13
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	ISPS, Program Verification, Microcode Correctness, State Delta Verification System, Program Correctness

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report describes the progress made in the State Delta Verification System (SDVS) research and development effort for fiscal year 1986.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code) 22c. OFFICE SYMBOL

CONTENTS

I. INTRODUCTION

This report describes the progress made in the State Delta Verification System (SDVS) research and development effort for fiscal year 1986.

The long-term goal of the field of program verification is to be able to prove the correctness of any program with respect to its (correct) specification. Existing program verification systems can be classified according to their intended domain of application, degree of user interaction, and choice of specification language.

SDVS 5 is the current version of a program verification project that essentially started in the thesis of S. Crocker and has progressed since then in capability of proof, ease of user interface, and complexity of applications actually tackled. SDVS is a highly interactive proof checker for proofs of microcode correctness. SDVS 5 runs on the Symbolics Lisp Machine version 6. It consists of three modules: the kernel and user interface (6000 lines of lisp code), the simplifier (14,000 lines of lisp code), and the translator from the ISPS machine description language to state deltas (4000 lines of lisp code). The kernel directs the symbolic execution and checks the validity of the dynamic proof commands. The simplifier encodes the system's knowledge of static domains and allows the user to create and prove lemmas for future use. The translator converts descriptions written in an expanded ISPS to the internal state delta language.

We have successfully analyzed (proved correct or found bugs in) all but 4 of the 128 macroinstructions not involving I-O, maintenance, or diagnostics of the BBN C30 computer, covering approximately 1000 lines of microcode. Details of this work may be found in three papers by J. Cook.[2,3,4]

SDVS may be operated in interactive or batch mode. In batch mode, the typical method for checking a proof of correctness of implementation between a host and target level is to input descriptions of the host and target, the mapping between the two (relating the registers in the target to registers in the host), and the proof (written in the SDVS proof language). SDVS then checks the proof and returns a trace of the proof with diagnostic error messages in the case of incorrect or incomplete proof.

The use of the system is detailed in the users' manual.[5]

The SDVS project depends on the synergy among the three efforts of research, system development, and applications. The research effort is needed to justify the operation of the implemented SDVS system, i.e., that it leads to valid conclusions, and to chart new directions for implementation. The applications, in addition to the intrinsic value of knowing that a given program is correct, are needed to test the design of SDVS against real world objects and to motivate research directions.

SUMMARY OF THIS YEAR'S PROGRESS:

RESEARCH: The theory of state deltas was proved to be decidable by interpreting state deltas in the theory of linear order with unary predicates, which is known to be decidable. However, because of the complexity of the decision procedure, this is unlikely to have practical application.

The concept of read protection was formalized in model-theoretic semantics, and a set of axioms was given. This

set is not complete, but the theory is decidable. This theory may have application in the specification and verification of security constraints.

DEVELOPMENT: SDVS 5 now exists in very usable form on the Symbolics lisp machine. It allows for interactive or batch proofs of implementation correctness starting with extended ISPS descriptions of host and target. The extension to ISPS allows state deltas and assumptions to be interspersed in the description. There is a sophisticated user interface which utilizes the special features of the Symbolics machine.

We hope that by the end of next year we will have a "production quality" system.

II. RESEARCH

A. SEMANTICS OF READ

The research summarized in this section is described in detail in a paper by Marcus and Redmond.[6]

We have given a formal definition of the concept of a register being read during the execution of a process. This definition may be given in the language of pure model theory. A candidate definition for secure implementation of target spec by host code is proposed.

We give some axioms for read protection, but do not know how to make this set into a complete axiomatization. The theory of read is proved decidable (in the naturally restricted cases).

An introduction to the main idea of the formalization follows: In dealing with computer security, a major consideration is protecting certain registers from being read from or written into by unauthorized processors or users. It is fairly trivial to define the semantics of write: if the value of the register changes, it was written into. (This is ignoring the rare occasion when writing a value which is the same as the current value may be of interest.)

However, detecting when a register's value has been read is a much more delicate matter. For a register x to be read, we do not require that the reader actually "look at" or access the x, nor that the reader learn the value of the contents of x in any way. We view "reading" as a special case of the general problem of information flow. Intuitively, we will consider a register x to have been read by (or during) process P, if some non-public or protected information about its contents becomes known during an execution of P, i.e., if the behavior of P is dependent on the value of x in some specifiable or observable way. This means that the concept of "non-public" must be made explicit in every specific case of read.

A superficial approach to the semantics of read yields the following examples. If the right-hand side of an assignment statement consists of the program variable x by itself, then x is read. If the expression x – x (subtraction) is on the right-hand side, it is not completely clear whether we want to consider x to have been read or not. If x appears in a condition for a branch, where the outcome of branch depends on the value of x, we probably do want to consider that x has been read, even if we don't need to know its explicit value.

These examples all rely on the presence of some syntax for their formulation. The situation is clearly different in the case of security verification. We shall consider a prototypical relation between an adversary, A, and a process, P. The adversary tries to learn something new about x by examining the behavior of the process and by using the public knowledge, K, available about x. A, in general, does not have complete knowledge about the syntax of P; A may observe P in operation or may wait until P has terminated (if ever), and then analyze the results to deduce the new information about x. In this case, disclosure of new knowledge about x means that some behavior of x which is *a priori* possible in the context of of K, is discovered to be impossible in light of P.

The link between the syntactic and the general formulation can be seen, for example, in the simple assignment statement above: a possible value of x (actually, all but one possible value of x) is eliminated by examining the value of the left-hand side. Likewise in the branch example, the negation of the realized branch predicate is discovered to

be impossible at that point in time in that computation (the realized branch predicate is discovered to be true).

Our approach elevates this "public knowledge" to a position of prominence in the definition. K plays a dual role in the sense that it can be used by the adversary to deduce information about x based on observation of P, but K also is the criterion for deciding if that information about x is new. For example, if the public knowledge about x is weak (e.g., K = TRUE), and x is not an explicit variable of P, then x is not read by P because there is no way to connect x with the behavior of P. Also, if K is too strong (e.g., K = FALSE), then x is not read by P, because P could not possibly increase our knowledge about x. Actually, x can alternate between being read and not being read with respect to K as K increases in strength. On the other hand, for any non-trivial process P and register x, there is some public knowledge such that with respect to it, P reads x: simply take K to be x = v for some variable, v, of P.

We assume a strict distinction on the one hand between the variables of a process, Var(P), which P knows everything about at all points of all computations and by definition reads, analogous to real locations in a machine for P, and on the other hand, external variables which P may or may not read, depending on K.

We take the view that the specification of a process is a way of restricting the possible computations that the user can perform, rather than permitting them. We are interested in protecting against the inadvertent read, not finding which registers are always read.

There are four separable concerns which need formalization: the new information learned about x by P is "information", it is "new", it is "about" x, and it is learned "by" P. As mentioned above, the "newness" will be measured in relation to K; "information" is taken to be a set of possible computations. The "about x" and "by P" are handled by looking at the restriction of a model of K to x, and combining this with a computation of P.

To formalize the above discussion in precise mathematics, we utilize the concepts of model theory. We start out with a model (computation) of K, and we restrict it to x (ignore the other variables). This represents a possible behavior of (or information about) x consistent with K. Now take a model (computation) of P and see if we can superimpose the above restricted model onto this model of P in a way which is consistent with K, i.e, such that the combination is a model of K. If we cannot, then we have learned that this behavior of x is ruled out by this particular computation of P, and x is read. It could be that the forbidden behavior (the information) is specifiable by a sentence in a given language. This means that there is a sentence, F, such that the above holds for all models of $K \wedge \neg F$. (In other cases, it may be that a particular model or set of models is ruled out, but this model or set of models is not specifiable in the language.) If M does not read x with respect to K, then the adversary cannot deduce anything about x that does not already follow from K.

We examine the possibility of expressing the necessary semantics within various temporal logics and come to the conclusion that this is impossible in some cases.

There are several possible variations of the formalization which seem reasonable. An important task is to examine examples and results about their interdependencies in order to determine which ones correctly represent our intuition.

We give a set of axioms for read, but do not yet know how to get a complete axiomatization. The theory of read

(with some natural restrictions) is proved decidable by reducing to Rabin's S2S theorem.

B. STATE DELTAS

The theory of state deltas was proved decidable by using the decidability of the theory of linear order with unary predicates. This is a "theoretical" result, with little chance of practical applicability in SDVS.

III. DEVELOPMENT

SDVS 5 is the new version of the State Delta Verification System on the Symbolics lisp machine. It is a very usable tool for microcode verification, either in interactive mode or batch mode. The interactive mode utilizes some of the fancy Symbolics features and has on-line help, editing, and system prompts. There are (rudimentary) capabilities for proof manipulation and system querying, in addition to the imperative proof commands. The translator converts ISPS extended by mark points, assertions, assumptions, and state deltas into pure state deltas. The translation can be done for two "granularities": considering every ISPS statement to be significant, or only considering the effects between mark points. There are facilities for declaring and loading large arrays of data (e.g., the ROM contents). Axiom lists may be loaded and examined by name or content (axioms dealing with a given list of symbols). Lemmas may be created, loaded, proved, stored, and used.

The development of SDVS has been pushed and tested by a rather large real example, the microcoded C30 IMP machine.

The following sections describe some of the new features of SDVS 5. For more details and examples see the manual.[5]

A. USER INTERFACE

The user interface and its development are described in detail in a report by Landauer, Cohen, and Redmond.[7] *Here we list several highlights of the new version.*

1) The "SDVS-listener", which is the user environment, is invoked by select-s on the Symbolics. Dribbling (recording the session on a file) is accomplished by the **dribble-on** and **dribble-off** commands.

2) The user may input proof commands and expressions in the output infix (prettyprint) language.

3) Prompting for unspecified arguments: all commands prompt the user with the next argument name needed.

4) On-line help: there is an extensive help facility which uses the built-in Symbolics help utility.

5) Numbering of proof steps: proof steps are now numbered in a tree structure labeling scheme.

6) SDVS gives the user a filename string to edit. For example, if you wanted to execute command "isps" you'd see this:

```
<sdvs.1> ISPS pathname: mr.p:>eve>foo.isp
```

where you've typed in just the command isps. The initial default is foo.isp in the user's home directory. You could then edit the pathname, and whatever value is eventually accepted becomes the new default. The following defaults are available (and can be set by the user)

```
*default-isps-pathname* : "{home-dir}>foo.isp"
 used in isps, mpisps, tr, mptr
*default-axiom-pathname* : "sdvs:axioms;axioms.axioms"
 used in readaxioms, writeaxioms
*default-lemma-pathname* : "sdvs:lemmas;lemmas.lemmas"
 used in readlemmas, writelemmas
*default-isps-filename* : foo.isp
 used for state delta names (e.g., isps(foo.isp)).
```

7) **redo** expr, where expr is either a proof command name or a proof step number will redo that command.

8) **date** inserts the date comment-style into the proof, and on rerunning the proof, the old date is ignored and a new one used.

9) **setflag** is to be used to set on-off type flags (e.g., autoclose). The default value is the flip of the current value.

10) **setlevel** is to be used to set abbreviationlevel and **unique-name-level**. These last two indicators can now have either numerical values or nil. If the value is numerical, it will be used silently as the default, as in isps or ppsd. If it is NIL, you will be prompted for the value to use (the defaults being 0 and 1 respectively).

11) **whynotgoal** takes a simp option.

12) Messages that appear upon completion of a proof have been improved, in particular with respect to **defer**.

B. PROOF CAPABILITY

A detailed description of new developments in the bitstring solver appear in a paper of T. Redmond.[8]

We mention some other areas.

1) Two new self-explanatory query commands are **sdtobeproven** and **lastappliedsd**.

2) It is now true that if one binds an atom-name to a list of formulas and then puts the expression **formulas**(atom-name) in the pre- or postcondition of a state delta, each of the formulas in the list get inserted into the pre- or postcondition of the state delta. **Formulas** interprets the formulas listed under the atom-name as follows: if the formula is an atom, then it is checked to see if that atom is attached to a state delta. If so, the state delta is inserted; otherwise the atom is inserted as a predicate. Otherwise the formula is inserted.

3) The level-to-level mapping has been generalized to allow a weak universal quantifier, that is, the capability to say that for each index in a given range, an array with that index maps to corresponding elements in another array. The syntax is "(map hmem[i:j] (tmem[k:l]) (forall n i j f(tmem[n+k-i])))", where f is some function, such as a specified substring. This has the effect of mapping hmem[n] onto f(tmem[n+k-i]) for all n from i to j. At invocation time, a check is made that l-k = j-i.

4) **hidepropagations** and **restorepropagations** installed; these commands allow you to ignore conditional facts if they are not needed.

5) Selecti: The **selecti** command (the "i" stands for integer) allows the user to identify the next proof step (or sequence of proof steps) quickly, typically which state delta to apply, thereby shortcircuiting a perhaps lengthy check of usable state deltas. The state delta chosen to be applied is then, of course, processed in the usual fashion, checking for applicability, etc. The formal syntax is

(selecti <integer-term>
 (<range$_1$> . proofcommands$_1$)
 (<range$_2$> . proofcommands$_2$)
 ...
 (<range$_k$> . proofcommands$_k$)
 [(t . proofcommands$_{otherwise}$)]
)

where the ranges are ISPS range expressions, i.e., a number, a sequence of numbers, or an integer interval. This method is faster than pure symbolic execution, but we have no exact figures for comparison yet.

6) Compose: The **compose n** command, now only partially implemented, will allow the user to form one cumulative state delta out of the past n state deltas which were applied. This command is used now only in the case of straight line symbolic execution. Careful usage of the **selecti** and **compose** commands can greatly speed up a proof.

7) Defer: The proof command **defer** allows the user to assume one or more of the current goals to be true, and then move on to the next part of the proof. This is a partial implementation of a command which would allow the user to move arbitrarily from one part of a proof to another, and is useful in building up a proof interactively. The user has the option of deferring all the current goals, or only a set of named goals. The names of the current goals are listed by the **whynotgoal** query. At the end of a proof with deferred goals, a comment is attached stating how many goals have been deferred.

8) Popping: The successfully completed proof steps are recorded and labeled in the proof stack. The state of the system at the end of each proof step is saved, so that the user may return, via the **pop** command, to the state at any label. The states under that label are thereby eliminated from the proof stack and lost. However, when popping is finally implemented correctly, the user will be able to rerun those proof steps beneath the destination of the pop. Then **pop** and **defer** will be a useful pair of commands.

9) A new set of axioms was written to facilitate proofs dealing with substrings of "exclusive or" and other bitstring logical operations.

10) The simplifier recognizes and utilizes (in certain cases) bitstrings with constant substrings.

C. INTERACTION WITH ISPS

Much valuable experience about the specification power and naturalness of ISPS has been gained by the C/30 work. These lessons are explained in the technical report.[9]

1) ISPS has been extended to allow the insertion of "assertions", "assumptions", and general state deltas in the code. (These additional expressions were formerly called "annotations".) Assertions are just static state deltas, while assumptions translate to preconditions of state deltas which essentially contain the whole rest of the ISPS code as the postcondition.

2) MPISPS (TR between mark points) can be used in the precondition of a state delta. The typical use for this facility is to express the target machine of an implementation problem in a granularity coarser than every ISPS state change.

REFERENCES

1. S. D. Crocker, *State Deltas: A Formalism for Representing Segments of Computation*, PhD dissertation, University of California, Los Angeles, 1977.

2. J. V. Cook, "Final Report for the C/30 Microcode Verification Project", Tech. report TR-0086(6771)-4, The Aerospace Corporation, September 1986.

3. J. V. Cook, "C/30 Proof", Tech. report TR-0086(6771)-5, The Aerospace Corporation, September 1986.

4. J. V. Cook, "Proof Strategy for the Verificaton of the C/30 Microcode", Tech. report TR-0086(6771)-3, The Aerospace Corporation, September 1986.

5. L. Marcus, "SDVS 5 Users' Manual", Tech. report TR-0086(6778)-2, The Aerospace Corporation, 1986.

6. L. Marcus and T. Redmond, "A Semantics of Read", Tech. report ATR-86(8454)-1, The Aerospace Corporation, 1986, Also in Proceedings of 9th National Computer Security Conference, p. 184-193

7. J. Landauer, T. Redmond, and E. Cohen, "The SDVS User Interface", Tech. report TR-0086(6778)-4, The Aerospace Corporation, 1986.

8. T. Redmond, "Status Report on the Bitstring Solver", Tech. report TR-0086(6778)-3, The Aerospace Corporation, 1986.

9. E. Cohen and J. Landauer, "Specification Problems Encountered during the Proof of the C/30 Microcode", Tech. report TR-0086(6778)-5, The Aerospace Corporation, 1986.

LABORATORY OPERATIONS

The Aerospace Corporation functions as an "architect-engineer" for national security projects, specializing in advanced military space systems. Providing research support, the corporation's Laboratory Operations conducts experimental and theoretical investigations that focus on the application of scientific and technical advances to such systems. Vital to the success of these investigations is the technical staff's wide-ranging expertise and its ability to stay current with new developments. This expertise is enhanced by a research program aimed at dealing with the many problems associated with rapidly evolving space systems. Contributing their capabilities to the research effort are these individual laboratories:

Aerophysics Laboratory: Launch vehicle and reentry fluid mechanics, heat transfer and flight dynamics; chemical and electric propulsion, propellant chemistry, chemical dynamics, environmental chemistry, trace detection; spacecraft structural mechanics, contamination, thermal and structural control; high temperature thermomechanics, gas kinetics and radiation; cw and pulsed chemical and excimer laser development including chemical kinetics, spectroscopy, optical resonators, beam control, atmospheric propagation, laser effects and countermeasures.

Chemistry and Physics Laboratory: Atmospheric chemical reactions, atmospheric optics, light scattering, state-specific chemical reactions and radiative signatures of missile plumes, sensor out-of-field-of-view rejection, applied laser spectroscopy, laser chemistry, laser optoelectronics, solar cell physics, battery electrochemistry, space vacuum and radiation effects on materials, lubrication and surface phenomena, thermionic emission, photosensitive materials and detectors, atomic frequency standards, and environmental chemistry.

Computer Science Laboratory: Program verification, program translation, performance-sensitive system design, distributed architectures for spaceborne computers, fault-tolerant computer systems, artificial intelligence, microelectronics applications, communication protocols, and computer security.

Electronics Research Laboratory: Microelectronics, solid-state device physics, compound semiconductors, radiation hardening; electro-optics, quantum electronics, solid-state lasers, optical propagation and communications; microwave semiconductor devices, microwave/millimeter wave measurements, diagnostics and radiometry, microwave/millimeter wave thermionic devices; atomic time and frequency standards; antennas, rf systems, electromagnetic propagation phenomena, space communication systems.

Materials Sciences Laboratory: Development of new materials: metals, alloys, ceramics, polymers and their composites, and new forms of carbon; nondestructive evaluation, component failure analysis and reliability; fracture mechanics and stress corrosion; analysis and evaluation of materials at cryogenic and elevated temperatures as well as in space and enemy-induced environments.

Space Sciences Laboratory: Magnetospheric, auroral and cosmic ray physics, wave-particle interactions, magnetospheric plasma waves; atmospheric and ionospheric physics, density and composition of the upper atmosphere, remote sensing using atmospheric radiation; solar physics, infrared astronomy, infrared signature analysis; effects of solar activity, magnetic storms and nuclear explosions on the earth's atmosphere, ionosphere and magnetosphere; effects of electromagnetic and particulate radiations on space systems; space instrumentation.

END

10-87

DTIC