Abstracts of Reports and Papers Available by Anonymous FTP

Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Librarian: Judith Burgess
Burgess@csl.sri.com
Phone: +1 (415) 859-5924 Fax: +1 (415) 859-2844

Last Updated February 14, 1995

Contents

How to retrieve a copy of a paper or report

Reports and papers by members of the Formal Methods Program of SRI's Computer Science Laboratory can be obtained by anonymous ftp from `ftp.csl.sri.com` (`192.12.33.94`) in directory `/pub/reports`. To retrieve a file do the following

```
ftp ftp.csl.sri.com
 login: anonymous (or ftp)
 password: <your net-address>
cd pub/reports
binary
get <file-concerned>
quit
```

The following pages give abstracts for the papers available; in the heading to each abstract is a name such as `csl-91-3`. The files are stored in the form of LaTeX `.dvi` files (some are also available in postscript—see later) that have been compressed by the Unix `compress` program, and the corresponding file name is formed by appending `.dvi.Z` to the abstract name, to yield, for example, `csl-91-3.dvi.Z`. Some more recent files are compressed with the Free Software Foundation's `gzip` program, these files have names such as `csl-91-3.dvi.gz`. If your system lacks the `gunzip` program, ask your system administrator to obtain it for you.

Once you have obtained a compressed file, you must uncompress it by:

```
uncompress csl-91-3.dvi.Z
```

(or on some machines by `compress -d csl-91-3.dvi.Z`), or by

```
gunzip csl-91-3.dvi.gz
```

(or on some machines by `gzip -d csl-91-3.dvi.gz`).

This will leave you with the file `csl-91-3.dvi`, which you can then print using whatever dvi-to-printer programs are available at your site. Some of the dvi files use "tpic specials" or landscape orientation (in figures), or unusual fonts; if these give trouble, first look in the directory `/pub/reports/postscript` to see if there is a postscript version available. If not, or if you cannot print postscript, let us know and we will try to create a version that works for you. (Also, suggest to your system administrators that they consider getting a recent version of `dvips` and other TeX/LaTeX printing and previewing programs).

As noted, some of the files are also available in postscript form in the directory `/pub/reports/postscript`. A few that contain complicated graphics are available *only* in postscript—these are noted in the abstracts below. Postscript files have names of the form `<name>.ps.Z` or `<name>.ps.gz`. Obtain and uncompress these files as described above, then send the resulting `<name>.ps` file to a postscript printer.

An alternative way to access these files is through world-wide-web to `http://www.scl.sri.com/sri-csl-fm.html`.

1 File aaai88

Position Paper for AAAI-88 Workshop on V&V for Knowledge-Based Systems, Reprinted in Uma G. Gupta, editor. *Validating and Verifying Knowledge-Based Systems*, pp. 76–82. IEEE Computer Society, Los Alamitos, CA, 1991.

Validation and Testing of Knowledge-Based Systems
How bad can it get?

John Rushby
Computer Science Laboratory
SRI International

Short paper: no abstract provided.

2 File aaai91

Presented at AAAI-91, July 14-19, Anaheim, CA and appears in *Proceedings, AAAI-91, Volume 2*, pages 836-841.

Model-Based Reconfiguration: Toward an Integration with Diagnosis

Judith Crow and John Rushby
Computer Science Laboratory
SRI International
Menlo Park, California 94025 USA
crow@csl.sri.com, rushby@csl.sri.com

We extend Reiter's general theory of model-based diagnosis[1] to a theory of reconfiguration. The generality of Reiter's theory readily supports an extension in which the problem of reconfiguration is viewed as a close analogue of the problem of diagnosis. Using a reconfiguration predicate **rcfg** analogous to the abnormality predicate **ab**, we formulate a strategy for reconfiguration by transforming that for diagnosis. A benefit of this approach is that algorithms for diagnosis can be exploited as algorithms for reconfiguration, thereby promoting an integrated approach to fault detection, identification, and reconfiguration.

[1] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, April 1987.

3 File aamp5 (available in postscript only)

Submitted for publication

Formal Verification of the AAMP5 Microprocessor: A Case Study in the Industrial Use of Formal Methods

Steven P. Miller
Collins Commercial Avionics
Rockwell International
Cedar Rapids IA 52498 USA

Mandayam Srivas
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

This paper describes the experiences of Collins Commercial Avionics and SRI International in formally specifying and verifying the microcode for the AAMP5 microprocessor with the PVS verification system. The project was conducted to determine if an industrial microprocessor designed for use in real-time embedded systems could be formally specified at the instruction set and register transfer levels and if formal proofs could be used to prove the microcode correct. The paper provides a brief technical overview, but its emphasis is on the lessons learned in using PVS on an example of this size and the implications for using formal methods in an industrial setting.

4 File cade88

Appeared in CADE 9, 1988, LNCS 310.

Adventures in Associative-Commutative Unification

Patrick Lincoln and Jim Christian
MCC, Systems Languages
3500 W. Balcones Cntr. Dr.
Austin, TX, 78759

We have discovered an efficient algorithm for matching and unification in associative-commutative (AC) and associative-commutative-idempotent (ACI) equational theories. In most cases of AC unification and in all cases of ACI unification our method obviates the need for solving diophantine equations, and thus avoids one of the bottlenecks of other associative-commutative unification techniques. The algorithm efficiently utilizes powerful constraints to eliminate much of the search involved in generating valid substitutions. Moreover, it is able to generate solutions lazily, enabling its use in an SLD-resolution-based environment like Prolog. We have found the method to run much faster and use less space than other associative-commutative unification procedures on many commonly encountered AC problems.

5 File cade92-ns

Appeared in 11th Conference on Automated Deduction, Saratoga Springs, NY.

Proof Search in the Intuitionistic Sequent Calculus

N. Shankar
SRI International Computer Science Laboratory
Menlo Park, CA 94025 USA

The use of Herbrand functions (sometimes called *Skolemization*) plays an important role in classical theorem proving and logic programming. We define a notion of Herbrand functions for the full intuitionistic predicate calculus. This definition is based on the view that the proof-theoretic role of Herbrand functions (to replace universal quantifiers), and of unification (to find instances corresponding to existential quantifiers), is to ensure the eigenvariable conditions on a sequent proof. The propositional impermutabilities that arise in the intuitionistic but not the classical sequent calculus motivate a generalization of the classical notion of Herbrand functions. This generalization of Herbrand functions also applies to the sequent calculus formalizations of logics other than intuitionistic predicate calculus.

6 File cade92-pvs

Reprint from Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, Saratoga, NY, 1992, pages 748–752; Volume 607 of Springer Verlag *Lecture Notes in Artificial Intelligence*.

PVS: A Prototype Verification System

S. Owre, J. M. Rushby, and N. Shankar
SRI International Computer Science Laboratory
Menlo Park, CA 94025 USA

Short paper: no abstract provided.

See also **pvs-language**, **pvs-prover**, **pvs-system**, **pvs-tutorial**, and **pvs-refcard**. The PVS system is available in directory **/pub/pvs**.

7 File cav93-hybrid

Reprint from *Computer Aided Verification, CAV '93,* Costas Courcoubetis, editor, Elounda, Greece, June/July 1993. Volume 697 of Springer Verlag *Lecture Notes in Computer Science*, pages 292–304

The Formal Verification of an Algorithm for Interactive Consistency under a Hybrid Fault Model

Patrick Lincoln and John Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Modern verification systems such as PVS are now reaching the stage of development where the formal verification of critical algorithms is feasible with reasonable effort. This paper describes one such verification in the field of fault tolerance. The distribution of single-source data to replicated computing channels (Interactive Consistency or Byzantine Agreement) is a central problem in this field. The classic Oral Messages (OM) algorithm solves this problem under the assumption that all channels are either nonfaulty or arbitrarily (Byzantine) faulty. Thambidurai and Park have introduced a "hybrid" fault model that distinguishes additional fault modes, along with a modified version of OM. They gave an informal proof that their algorithm withstands the same number of arbitrary faults, but more "nonmalicious" faults than OM.

We detected a flaw in this algorithm while undertaking its formal verification using PVS. The discipline of mechanically-checked formal verification helped us to develop a corrected version of the algorithm. Here we describe the formal specification and verification of this new algorithm. We argue that formal verification systems such as PVS are now sufficiently effective that their application to critical fault-tolerance algorithms should be considered routine.

8 File cav93-realtime

Reprint from *Computer Aided Verification, CAV '93,* Costas Courcoubetis, editor, Elounda, Greece, June/July 1993. Volume 697 of Springer Verlag *Lecture Notes in Computer Science*, pages 280–291.

Verification of Real-Time Systems Using PVS

N. Shankar
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

We present an approach to the verification of the real-time behavior of concurrent programs and describe its mechanization using the PVS proof checker. Our approach to real-time behavior extends previous verification techniques for concurrent programs by introducing a new operator for reasoning about absolute time. This operator returns the amount of time that has elapsed since the given predicate last held in the current computation. This approach is mechanized by first encoding the computational behavior of real-time systems within the higher-order logic of PVS and employing the interactive proof checker of PVS to develop the proofs. Our approach is illustrated using two examples: Fischer's real-time mutual exclusion protocol and a railroad crossing controller. We also discuss related approaches to formalizing real-time behavior.

9 File cav94 (available in postscript only)

Reprint from David Dill Ed., *Computer-Aided Verification*, Stanford, CA, June 1994. Springer Verlag LNCS 818, pp. 247–259.

Ground Temporal Logic: A Logic for Hardware Verification

David Cyrluk
Dept. of Computer Science, Stanford University, Stanford CA 94305 and
Computer Science Laboratory, SRI International, Menlo Park, CA 94025
cyrluk@cs.stanford.edu

and

Paliath Narendran
SUNY-ALBANY, Albany NY
dran@cs.albany.edu

We present a new temporal logic, *GTL*, appropriate for specifying properties of hardware at the register transfer level. We argue that this logic represents an improvement over model checking for some natural hardware verification problems. We show that the validity problem for this logic is Π^1_1 complete. We then identify a fragment of the logic that is decidable. We show that in this fragment we are still able to encode many interesting problems, including the correctness of pipelined microprocessors.

10 File compass94

From Compass '94 (Proceedings of the Ninth Annual Conference on Computer Assurance),IEEE, Gaithersburg, June 1994, pp. 107–120.

Formal Verification of an Interactive Consistency Algorithm for the Draper FTP Architecture Under a Hybrid Fault Model

Patrick Lincoln and John Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Fault-tolerant systems for critical applications should tolerate as many *kinds* of faults and as large a *number* of faults as possible, while using as little hardware as feasible. And they should be provided with strong assurances for their correctness.

Byzantine fault-tolerant architectures are attractive because they tolerate any kind fault, but they are rather expensive: at least $3m + 1$ processors are required to withstand m arbitrary faults. Two recent developments mitigate some of the costs: algorithms that operate under a *hybrid fault model* tolerate more faults for a given number of processors than classical Byzantine fault-tolerant algorithms, and *asymmetric architectures* tolerate a given number of faults with less hardware than conventional architectures. In this paper we combine these two developments and present an algorithm for achieving interactive consistency (the problem of distributing sensor samples consistently in the presence of faults) under a hybrid fault model on an asymmetric architecture.

The extended fault model and asymmetric architecture complicate the arguments for the correctness and the number of faults tolerated by the algorithm. To increase assurance, we have formally verified these properties and checked the proofs mechanically using the PVS verification system. We argue that mechanically-supported formal methods allow for effective reuse of intellectual resources, such as specifications and proofs, and that exercises such as this can now be performed very economically.

11 File compcon91

Appeared in Proceedings of IEEE Compcon, San Francsico CA, February 1991, pp. 450–455.

From Formal Verification to Silicon Compilation

J. Joyce[1], E. Liu, J. Rushby, N. Shankar, R. Suaya, F. von Henke

Computer Science Laboratory, SRI International
333 Ravenswood Ave., Menlo Park, California 94025

Formal verification is emerging as a viable method for increasing design assurance for VLSI circuits. Potential benefits include reduction in the time and costs associated with testing and redesign, improved documentation and ease of modification, and greater confidence in the quality of the final product. This paper reports on an experiment whose main purpose was to identify the difficulties of integrating formal verification with conventional VLSI CAD methodology. Our main conclusion is that the most effective use of formal hardware verification will be at the higher levels of VLSI system design, with lower levels best handled by conventional VLSI CAD tools.

[1]Department of Computer Science, University of British Columbia, 6356 Agricultural Road, Vancouver, B.C., Canada V6T 1W5

12 File constructive-reals

Bishop's Constructive Reals in EHDM

Sam Owre and John Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

We specify Errett Bishop's constructive definition of the reals in EHDM, and verify that the corresponding definition of addition is closed.

13 File csl-88-7r

Quality Measures and Assurance for AI Software

John Rushby

Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

Technical Report CSL-88-7R, September 1988
(Also available as NASA Contractor Report 4187)

This report is concerned with the application of software quality and assurance techniques to AI software. It describes work performed for the National Aeronautics and Space Administration under contract NAS1 17067 (Task 5) and is also available as a NASA Contractor Report.

The report is divided into three parts. In Part I we review existing software quality assurance measures and techniques–those that have been developed for, and applied to, conventional software. This part, which provides a fairly comprehensive overview of software reliability and metrics, static and dynamic testing, and formal specification and verification, may be of interest to those unconcerned with AI software. In Part II, we consider the characteristics of AI-based software, the applicability and potential utility of measures and techniques identified in the first part, and we review those few methods that have been developed specifically for AI-based software. In Part III of this report, we present our assessment and recommendations for the further exploration of this important area. An extensive bibliography with 194 entries is provided.

14 File csl-89-3r

Formal Verification of the Interactive Convergence Clock Synchronization Algorithm

John Rushby and Friedrich von Henke

Computer Science Laboratory
SRI International Menlo Park CA 94025 USA

Technical Report CSL-89-3R
February 1989, Revised August 1991
(Unrevised version also available as NASA Contractor Report 4239)

We describe a formal specification and mechanically checked verification of the Interactive Convergence Clock Synchronization Algorithm of Lamport and Melliar-Smith[1]. In the course of this work, we discovered several technical flaws in the analysis given by Lamport and Melliar-Smith, even though their presentation is unusually precise and detailed. As far as we know, these flaws (affecting the main theorem and four of its five lemmas) were not detected by the "social process" of informal peer scrutiny to which the paper has been subjected since its publication. We discuss the flaws in the published proof and give a revised presentation of the analysis that not only corrects the flaws in the original, but is also more precise and, we believe, easier to follow. This informal presentation was derived directly from our formal specification and verification. Some of our corrections to the flaws in the original require slight modifications to the assumptions underlying the algorithm and to the constraints on its parameters, and thus change the external specification of the algorithm.

The formal analysis of the Interactive Convergence Clock Synchronization Algorithm was performed using the EHDM formal specification and verification environment. This application of EHDM provides a demonstration of some of the capabilities of the system.

Note: This second edition of the report presents a revised version of the formal specification and verification that exploits some of the features introduced into EHDM since the original verification was performed, and also improves the substance of the verification in three respects.

[1] L. Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(1):52–78, January 1985.

15 File csl-90-9 (available in postscript only)

On Location: Points about Regions

Judith S. Crow
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA
Crow@csl.sri.com

Peter B. Ladkin
ICSI
Berkeley CA 94704
Ladkin@icsi.Berkeley.edu

We formalize Whitehead's construction for inducing point structures from region structures using a primitive relation of *connection* on regions, thereby providing a spatiotemporal analogue to the construction of temporal periods (points) from events found in Kamp and in van Benthem.

16 File csl-91-2

An Introduction to Formal Specification and Verification Using EHDM

John Rushby, Friedrich von Henke, and Sam Owre

Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park CA 94025 USA

Technical Report CSL-91-2
February 1991

This report is a tutorial on formal specification and verification using EHDM. The EHDM specification language is very expressive, based on a strongly-typed higher-order logic, enriched with elements of the Hoare (relational) calculus. The type system provides subtypes, dependent types, and certain forms of type-polymorphism. Modules are used to structure large specifications and support hierarchical development. The language has a complete formal semantic characterization and is supported by a fully mechanized specification and verification environment that has been used to develop large specifications and perform very hard formal verifications.

The tutorial uses simple examples to describe the EHDM language, methodology, and tools. The first examples illustrate the basic ideas of specification and theorem proving in EHDM. We then introduce the ideas of testing specifications, of horizontal and vertical hierarchy, and of consistency and conservative extension. Later chapters cover more advanced topics including subtypes, higher-order logic, proofs by induction, and program verification using Hoare logic. The tutorial is illustrated throughout with self-contained examples of EHDM specifications and proofs, all of which have been mechanically checked.

17 File csl-91-3

Formal Specification and Verification of a Fault-Masking and Transient-Recovery Model for Digital Flight-Control Systems

John Rushby

Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Technical Report CSL-91-3, June 1991
(Also available as NASA Contractor Report 4384)

We present a formal model for fault-masking and transient-recovery among the replicated computers of digital flight-control systems. We establish conditions under which majority voting causes the same commands to be sent to the actuators as those that would be sent by a single computer that suffers no failures. The model and its analysis have been subjected to formal specification and mechanically checked verification using the EHDM system.

Keywords: digital flight control systems, formal methods, formal specification and verification, proof checking, fault tolerance, transient faults, majority voting, modular redundancy

18 File csl-91-4

Mechanical Verification of a Schematic Protocol for Byzantine Fault Tolerant Clock Synchronization

Natarajan Shankar
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Technical Report SRI-CSL-91-04, June 1991

Schneider generalizes a number of protocols for Byzantine fault tolerant clock synchronization and presents a uniform proof for their correctness. We present a machine checked proof of this schematic protocol that revises some of the details in Schneider's original analysis. The verification was carried out with the Ehdm system developed at the SRI Computer Science Laboratory. The mechanically checked proofs include the verification that the *egocentric mean* function used in Lamport and Melliar-Smith's Interactive Convergence Algorithm satisfies the requirements of Schneider's protocol.

19 File csl-92-1

Formal Verification of an Oral Messages Algorithm for Interactive Consistency

John Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Technical Report CSL-92-01
July 1992
Also available as NASA Contractor Report 189704, October 1992

We describe the formal specification and verification of an algorithm for Interactive Consistency[1] based on the Oral Messages algorithm for Byzantine Agreement[2]. We compare our treatment with that of Bevier and Young,[3,4] who presented a formal specification and verification for a very similar algorithm. Unlike Bevier and Young, who observed that "the invariant maintained in the recursive subcases of the algorithm is significantly more complicated than is suggested by the published proof" and who found its formal verification "a fairly difficult exercise in mechanical theorem proving," our treatment is very close to the previously published analysis of the algorithm, and our formal specification and verification are straightforward.

This example illustrates how delicate choices in the formulation of a problem can have significant impact on the readability of its formal specification and on the tractability of its formal verification.

[1]M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[2]Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[3]W. R. Bevier and W. D. Young. The design and proof of correctness of a fault-tolerant circuit. Pages 243–260 in J. F. Meyer and R. D. Schlichting, editors. *Dependable Computing for Critical Applications—2*, Tucson, AZ, February 1991. Springer-Verlag, Wien, Austria, volume 6 of *Dependable Computing and Fault-Tolerant Systems*.

[4]William R. Bevier and William D. Young. Machine-checked proofs of the design and implementation of a fault-tolerant circuit. NASA contractor report 182099, NASA Langley Research Center, Hampton, VA, November 1990.

20 File csl-92-2

Noninterference, Transitivity, and Channel-Control Security Policies

John Rushby

Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Technical Report CSL-92-02
December 1992

We consider noninterference formulations of security policies[1] in which the "interferes" relation is intransitive. Such policies provide a formal basis for several real security concerns, such as channel control,[2,3] and assured pipelines[4]. We show that the appropriate formulation of noninterference for the intransitive case is that developed by Haigh and Young for "multidomain security" (MDS)[5]. We construct an "unwinding theorem" for intransitive polices and show that it differs significantly from that of Haigh and Young. We argue that their theorem is incorrect. An appendix presents a mechanically-checked formal specification and verification of our unwinding theorem.

We also consider the relationship between transitive and intransitive formulations of security. We show that the standard formulations of noninterference and unwinding correspond exactly to our intransitive formulations, specialized to the transitive case. We show that transitive polices are precisely the "multilevel security" (MLS) polices, and that any MLS secure system satisfies the conditions of the unwinding theorem.

In addition, we consider the relationship between noninterference formulations of security and access control formulations, and we identify the "reference monitor assumptions" that play a crucial role in establishing the soundness of access control implementations.

---

[1] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proceedings of the Symposium on Security and Privacy*, pages 11–20, Oakland, CA, April 1982. IEEE Computer Society.

[2] John Rushby. The design and verification of secure systems. In *8th ACM Symposium on Operating System Principles*, pages 12–21, Asilomar, CA, December 1981. (ACM *Operating Systems Review*, Vol. 15, No. 5).

[3] John Rushby. Verification of secure systems. Technical Report 166, Computing Laboratory, University of Newcastle upon Tyne, Newcastle upon Tyne, UK, August 1981.

[4] W. E. Boebert and R. Y. Kain. A practical alternative to hierarchical integrity policies. In *Proceedings 8th DoD/NBS Computer Security Initiative Conference*, pages 18–27, Gaithersburg, MD, September 1985.

[5] J. Thomas Haigh and William D. Young. Extending the noninterference version of MLS for SAT. *IEEE Transactions on Software Engineering*, SE-13(2):141–150, February 1987.

21 File csl-92-12

Mechanized Verification of Real-Time Systems Using PVS

N. Shankar
Computer Science Laboratory
SRI International
Menlo Park CA 94025

Technical Report CSL-94-10, November, 1992

Some important properties of certain safety-critical systems can be established only by a careful analysis of their real-time behavior. These real-time systems are typically designed to also meet functional requirements where the real-time behavior is irrelevant. A suitable verification framework for such systems must therefore be capable of coping with functional as well as real-time behavior. We present an approach to the verification of the real-time behavior of concurrent programs and describe its mechanization using the PVS proof checker. Our approach to real-time behavior extends previous verification techniques for concurrent programs by introducing a new operator for reasoning about absolute time. This operator returns the amount of time that has elapsed since the given predicate last held in the current computation. This approach is mechanized by first encoding the computational behavior of real-time systems within the higher-order logic of PVS and employing the interactive proof checker of PVS to develop the proofs. Our approach is illustrated using two examples: Fischer's real-time mutual exclusion protocol and a railroad-crossing controller. We also discuss related approaches to formalizing real-time behavior.

22 File csl-93-1

Original version published in *Reliability Engineering and System Safety*, Vol. 43, No. 2, pp. 189–219, 1994

Critical System Properties: Survey and Taxonomy

John Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Technical Report CSL-93-01, May 1993
Revised, February 1994

Computer systems are increasingly employed in circumstances where their failure (or even their correct operation, if they are built to flawed requirements) can have serious consequences.

There is a surprising diversity of opinion concerning the properties that such "critical systems" should possess, and the best methods to develop them. The *dependability* approach grew out of the tradition of ultra-reliable and fault-tolerant systems, while the *safety* approach grew out of the tradition of hazard analysis and system safety engineering. Yet another tradition is found in the *security* community, and there are further specialized approaches in the tradition of *real-time* systems. In this report, I examine the critical properties considered in each approach, and the techniques that have been developed to specify them and to ensure their satisfaction.

Since systems are now being constructed that must satisfy several of these critical system properties simultaneously, there is particular interest in the extent to which techniques from one tradition support or conflict with those of another, and in whether certain critical system properties are fundamentally compatible or incompatible with each other. As a step toward improved understanding of these issues, I suggest a taxonomy, based on Perrow's analysis[1], that considers the complexity of component interactions and tightness of coupling as primary factors.

[1]C. Perrow. *Normal Accidents: Living with High Risk Technologies.* Basic Books, New York, NY, 1984.

23 File csl-93-2

Also available as NASA Contractor Report 4527, July 1993

A Formally Verified Algorithm for Interactive Consistency Under a Hybrid Fault Model

Patrick Lincoln and John Rushby

Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Technical Report CSL-93-02
March 1993

Consistent distribution of single-source data to replicated computing channels is a fundamental problem in fault-tolerant system design. The "Oral Messages" (OM) algorithm of Lamport, Shostak, and Pease solves this problem of Interactive Consistency (also known as Source Congruence or Byzantine Agreement) in the presence of m arbitrary (i.e., Byzantine) Faults, using $m + 1$ rounds of message exchange and $n > 3m$ channels.

A problem with OM and similar algorithms is that all faults are assumed to exhibit worst-case (i.e., arbitrary) behavior, so that the algorithm can tolerate no more "simple" faults than truly Byzantine ones. To overcome this deficiency, Thambidurai and Park introduced a "hybrid" fault model that distinguished three fault modes: asymmetric (Byzantine), symmetric, and benign; they also exhibited, along with an informal "proof of correctness," a modified version of OM that withstands a asymmetric, s symmetric, and b benign faults simultaneously, using $m + 1$ rounds, provided $n > 2a + 2s + b + m$, and $m \geq a$.

Unfortunately, this algorithm is flawed; it fails, for example, in the case $n = 5, m = 1$ when the transmitter has a benign fault and one of the receivers is Byzantine. We detected this flaw while undertaking a formal verification of the algorithm using our PVS mechanical verification system. Repairing this algorithm is not easy. We developed an incorrect version ourselves, and even "proved" it correct using ordinary, informal mathematics.

The discipline of mechanically checked formal verification eventually enabled us to develop a correct algorithm for Interactive Consistency under the hybrid fault model. We present this algorithm, discuss its subtle points, and describe its formal specification and verification. We argue that formal verification systems such as PVS are now sufficiently effective that their application to fault-tolerance algorithms should be considered routine.

Keywords: interactive consistency, Byzantine agreement, hybrid fault models, formal verification.

24 File csl-93-4

Eight Papers on Formal Verification

Patrick Lincoln, Sam Owre, John Rushby,
Natarajan Shankar, and Friedrich von Henke

Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Technical Report SRI-CSL-93-04, May 1993

This volume collects together eight recent papers on the use and development of tools for formal specification and verification.

1. *Formal Verification for Fault-Tolerant Architectures: Some Lessons Learned*, by Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke summarizes the results and observations described in papers 2 through 7 below.
2. *Formal Verification of Algorithms for Critical Systems*, by John Rushby and Friedrich von Henke discusses the use of formal verification in applications such as digital flight-control systems, and describes verification of the Interactive Convergence Clock Synchronization Algorithm as an example.
3. *Mechanical Verification of a Schematic Byzantine Fault-Tolerant Clock Synchronization Algorithm*, by Natarajan Shankar describes the significantly difficult verification of a whole class of clock synchronization algorithms, that includes the Interactive Convergence Algorithm as a special case.
4. *Formal Specification and Verification of a Fault-Masking and Transient-Recovery Model for Digital Flight-Control Systems*, by John Rushby describes verification of the circumstances under which a majority-voting scheme allows replicated computers to recover from transients and to mask other faults.
5. *PVS: A Prototype Verification System*, by Sam Owre, John Rushby, and Natarajan Shankar describes our new verification system. PVS builds on the experience gained performing the verifications described in the papers listed above, and was used in the verifications described in the papers listed below.
6. *A Formally Verified Algorithm for Interactive Consistency Under a Hybrid Fault Model*, by Patrick Lincoln and John Rushby describes an algorithm for reaching agreement in the presence of faults. This algorithm, a variation on the "Oral Messages" algorithm, provides greater fault-tolerance than ordinary Byzantine-resilient algorithms. The paper also describes how formal verification helped detect a flaw in a previously published algorithm for this problem.
7. *The Formal Verification of an Algorithm for Interactive Consistency under a Hybrid Fault Model* by Patrick Lincoln and John Rushby describes the formal verification of the algorithm described in the previous paper.
8. *Verification of Real-Time Systems Using PVS* by Natarajan Shankar presents an approach to the verification of the real-time behavior of concurrent programs and describe its mechanization using PVS. The approach is illustrated by two examples: Fischer's real-time mutual exclusion protocol and a railroad crossing controller.

25 File csl-93-7

Also available as NASA Contractor Report 4551

Formal Methods and the Certification of Critical Systems

John Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

`Rushby@csl.sri.com`
Phone: +1 (415) 859-5456 Fax: +1 (415) 859-2844

Technical Report CSL-93-7
December 1993

This report was prepared to supplement a forthcoming chapter on formal methods in the FAA Digital Systems Validation Handbook[1]. Its purpose is to outline the technical basis for formal methods in computer science, to explain the use of formal methods in the specification and verification of software and hardware requirements, designs, and implementations, to identify the benefits, weaknesses, and difficulties in applying these methods to digital systems used in critical applications, and to suggest factors for consideration when formal methods are offered in support of certification.

The report assumes a serious interest in the engineering of critical systems, and a willingness to read occasional mathematical formulas and specialized terminology, but assumes no special background in formal logic or mathematical specification techniques. (An appendix provides a rapid introduction to formal logic for those to whom this topic is new.) The discussion should be accessible to most people with an engineering background. It may also be of use to those who develop or advocate formal methods and are interested in their use in support of certification for critical systems.

The particular focus of this report is computer systems used on-board aircraft. Some background on these systems is provided so that those concerned with critical computer systems in other contexts may be able to reinterpret the airplane-specific material to suit their own field.

[1] *Digital Systems Validation Handbook–Volume II.* Federal Aviation Administration Technical Center, Atlantic City, NJ, February 1989. DOT/FAA/CT-88/10. The chapter on Formal Methods will probably form part of Volume III.

26 File csl-93-8

A Lazy Approach to Compositional Verification

N. Shankar
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA
Phone: (415)859-5272 shankar@csl.sri.com

Technical Report CSL-93-08, December 1993

The verification of concurrent systems with a large number of interacting components poses a significant challenge to existing verification methodologies. We present a simple approach to compositional verification that allows global system properties to be established by considering only the behavior of the relevant system components. Each component specification describes the transitions due to the component itself and constrains the transitions due its environment. The composition of two specifications is an *interleaving conjunction* of the two specifications. Every property derived from a component specification remains true when this component is composed with other components. This compositional approach to verification is illustrated with the example of a buffer composed from two smaller buffers, and a real-time controller for a railroad crossing system composed from the subsystems for trains, the controller, and the gate. These examples have been verified using the PVS verification system.

27 File csl-93-9

Abstract Datatypes in PVS

N. Shankar
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA
Phone: (415)859-5272
shankar@csl.sri.com

Technical Report CSL-93-9
December 1993

PVS (Prototype Verification System) is a general-purpose environment for developing specifications and proofs. This document is one of a series of notes illustrating the pragmatics of using PVS. It deals primarily with the abstract datatype definition mechanism in PVS. The main example considered here is that of ordered binary trees. The binary tree data type is described as an abstract datatype in PVS. This datatype is defined to be parametric in its value type. The type of ordered binary trees is presented as a subtype of binary trees where the ordering relation is also taken as a parameter. We then define the operations of inserting an element into and searching for an element in an ordered binary tree. The bulk of the report is devoted to the PVS proofs of some useful properties of these operations. These proofs illustrate the various approaches to proving properties of abstract datatype operations. They also describe the various built-in capabilities of the PVS proof checker for simplifying abstract datatype expressions.

28 File csl-93-10

A Duration Calculus Proof Checker: Using PVS as a Semantic Framework

Jens U. Skakkebæk[1] & N. Shankar
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Technical Report CSL-93-10, December, 1993

The Duration Calculus (DC) is an interval temporal logic for reasoning about real-time systems. We present a Gentzen-style sequent calculus for DC and a tool for constructing DC specifications and checking DC proofs. This tool is implemented by encoding the semantics of DC within a general-purpose specification and verification environment called PVS. In encoding DC, we have used PVS as a semantic logical framework, that is, we have employed the higher-order logic of PVS to define the semantics of DC and we have formally derived the DC inference rules using this semantic definition. The DC inference rules can then be used without any knowledge of the semantic encoding. The inference capabilities of PVS such as rewriting, decision procedures, and user-defined proof strategies can be applied within DC proofs. The DC sequent calculus, the semantic encoding of DC in PVS, and the DC proof checking tool are illustrated using the example of a gas burner.

[1]Visiting SRI International Research Fellow. On leave from Department of Computer Science, Technical University of Denmark, DK-2800 Lyngby, Denmark.

29 File csl-93-12 (available in postscript only)

Microprocessor Verification in PVS

A Methodology and Simple Example

David Cyrluk
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Technical Report CSL-93-12, December, 1993

In the past we have verified a variety of different hardware circuits using different verification systems. In this paper we report on one such verification. The verification is that of a simple pipelined microprocessor first verified by Saxe, et. al.

We use this example as a vehicle for exploring a general methodology of verifying state machine systems in the PVS verification system. It is our hope that such a methodology can be extended to verify the correctness of actual commercial microprocessors.

30 File csl-94-10 (available in postscript only)

Transformations in High-Level Synthesis: Formal Specification and Efficient Mechanical Verification

P. Sreeranga Rajan[1]
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Technical Report CSL-94-10, October, 1994

Dependency graphs are used to model data and control flow in hardware and software design. In high-level synthesis of hardware, optimization and refinement transformations are used to transform dependency-graph-based specifications at the behavior level to dependency-graph-based implementations at the register-transfer level. Register-transfer-level implementations are mapped to gate-level hardware designs by low-level logic synthesis. In this work, we investigated the specification and mechanical verification of the correctness of transformations used in high-level synthesis of hardware.

We have provided a formal specification of dependency graphs, and verified the correctness of a variety of transformations used in an industrial synthesis framework. Errors have been discovered in the transformations, and modifications have been proposed and incorporated. Further, the formal specification has permitted us to examine the generalization and composition of transformations. In the process, we have discovered new transformations that could be used for further optimization and refinement than were possible before. The specification and verification schemes are general enough for applications in other synthesis frameworks and software design, where a transformational design approach is used.

In order to present our work in a concrete context, we focus on the high-level synthesis part of the SPRITE project at Philips Research Laboratories. The transformations in the high-level synthesis system are used for refinement and optimization of descriptions specified in a dependency graph language called the SPRITE Input Language (SIL). SIL is an intermediate language used during the synthesis of hardware described using languages such as VHDL, SILAGE and ELLA. Besides being an intermediate language, it forms the backbone of the TRADES synthesis system of the University of Twente. SIL has been used in the design of hardware for audio and video applications.

We used the Prototype Verification System (PVS) from SRI International to specify and mechanically verify the correctness of the transformations. The PVS specification language allows us to investigate the correctness problem using a convenient level of representation. The PVS verifier features automatic procedures and interactive verification rules to check properties of specifications.

---

[1]Visiting from Integrated Systems Design Laboratory, Department of Computer Science, University of British Columbia, Canada. Much of the work was done while visiting Philips Research Laboratories, Eindhoven, The Netherlands

31 File epri91

Presented at Electric Power Research Institute (EPRI) Workshop on Methodologies for Cost-Effective, Reliable Software Verification and Validation, Chicago, IL, 7–9 August 1991 and published in EPRI TR-100294, January 1992, pages 9-1 to 9-14. Reprinted in "*Readings in Ultra-Dependable Distributed Systems*," N. Suri, C. J. Walter and M. M. Hugue, Eds., IEEE Computer Society, 1994 (Forthcoming).

Formal Specification and Verification for Critical Systems: Tools, Achievements, and Prospects

John Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Formal specification and verification use mathematical techniques to help document, specify, design, analyze, or certify computer software and hardware. Mathematically-based notation can provide specifications that are precise and unambiguous and that can be checked mechanically for certain types of error. Formal verification uses theorem proving techniques to establish consistency between one level of formal specification and another.

This paper describes some of the issues in the design and use of formal specification languages and verification systems, outlines some examples of the application of formal methods to critical systems, and identifies the benefits that may be obtained from this technology.

32 File focs90

Appeared in FOCS 1990

Decision Problems for Propositional Linear Logic

Patrick Lincoln
Department of Computer Science, Stanford University, Stanford, CA 94305, and the Computer Science Laboratory, SRI International, Menlo Park, CA 94025

John Mitchell
Department of Computer Science, Stanford University, Stanford, CA 94305.

Andre Scedrov
Department of Mathematics, University of Pennsylvania, Philadelphia, PA 19104. Work completed while on sabbatical at the Computer Science Department and Center for the Study of Language and Information, Stanford University, Stanford, CA.

Natarajan Shankar
Computer Science Laboratory, SRI International, Menlo Park, CA 94025

Linear logic, introduced by Girard, is a refinement of classical logic with a natural, intrinsic accounting of resources. We show that unlike most other propositional (quantifier-free) logics, full propositional linear logic is undecidable. Further, we prove that without the modal *storage* operator, which indicates unboundedness of resources, the decision problem becomes PSPACE-complete. We also establish membership in NP for the multiplicative fragment, NP-completeness for the multiplicative fragment extended with unrestricted weakening, and undecidability for certain fragments of noncommutative propositional linear logic.

33 File fme93—Superseded by tse95

Reprint (slightly expanded) of a paper in *FME '93: Industrial-Strength Formal Methods*, pages 482–500, Odense, Denmark, April 1993. Volume 670 of Springer Verlag *Lecture Notes in Computer Science*.

Formal Verification for Fault-Tolerant Architectures: Some Lessons Learned

Sam Owre, John Rushby,
Natarajan Shankar, Friedrich von Henke[1]

Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

In collaboration with NASA's Langley Research Center, we are developing mechanically verified formal specifications for the fault-tolerant architecture, algorithms, and implementations of a "reliable computing platform" (RCP) for life-critical digital flight-control applications. To achieve a failure rate that is certifiably less than 10^{-9} per hour, RCP employs redundant, synchronized computing channels for fault masking and transient recovery. The synchronization and sensor-distribution algorithms are resilient with respect to a hybrid fault model that includes Byzantine faults.

Several of the formal specifications and verifications performed in support of RCP are individually of considerable complexity and difficulty. But in order to contribute to the larger goal, it has often been necessary to modify completed verifications to accommodate changed assumptions or requirements, and people other than the original developer have often needed to understand, build on, modify, or cannibalize an intricate verification.

Accordingly, we have been developing and honing our verification tools to better support these large, difficult, iterative, and collaborative verifications. Our goal is to reduce formal verifications as difficult as these to routine exercises, and to maximize the value obtained from formalization and verification.

In this paper, we describe some of the challenges we have faced, lessons learned, design decisions taken, and results obtained.

Note: Superseded by tse95

[1]Von Henke's main affiliation is now Fakultät für Informatik, Universität Ulm, Germany.

34 File ftcs93

Reprint from *Proceedings of the Fault-Tolerant Computing Symposium*, FTCS 23, Toulouse, France, June 1993, pp. 402–411

A Formally Verified Algorithm for Interactive Consistency Under a Hybrid Fault Model

Patrick Lincoln and John Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Thambidurai and Park[1] have proposed an algorithm for Interactive Consistency that retains resilience to the arbitrary (or Byzantine) fault mode, while tolerating more faults of simpler kinds than standard Byzantine-resilent algorithms. Unfortunately, and despite a published proof of correctness, their algorithm is flawed. We detected this while undertaking a formal verification of the algorithm.

We present a corrected algorithm that has been subjected to mechanically-checked formal verification. Because informal proofs seem unreliable in this domain, and the consequences of failure could be catastrophic, we believe formal verification should become standard for algorithms intended for safety-critical applications.

Keywords: Byzantine agreement, interactive consistency, hybrid fault models, formal verification.

[1]Philip Thambidurai and You-Keun Park. Interactive consistency with multiple failure modes. In *7th Symposium on Reliable Distributed Systems*, pages 93–100, Columbus, OH, October 1988. IEEE Computer Society.

35 File ftrtft92-jmr

Reprint from J. Vytopil, editor: *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Nijmegen, The Netherlands, January 1992, pages 237–257; Volume 571 of Springer Verlag *Lecture Notes in Computer Science*. Also available as Chapter 5 of J. Vytopil, editor: *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Kluwer International Series in Engineering and Computer Science, 1993.

Formal Specification and Verification of a Fault-Masking and Transient-Recovery Model for Digital Flight-Control Systems

John Rushby

Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

We present a formal model for fault-masking and transient-recovery among the replicated computers of digital flight-control systems. We establish conditions under which majority voting causes the same commands to be sent to the actuators as those that would be sent by a single computer that suffers no failures. The model and its analysis have been subjected to formal specification and mechanically checked verification using the EHDM system.

Keywords: digital flight control systems, formal methods, formal specification and verification, proof checking, fault tolerance, transient faults, majority voting, modular redundancy

36 File ftrtft92-ns

Reprint from J. Vytopil, editor: *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Nijmegen, The Netherlands, January 1992, pages 217–236; Volume 571 of Springer Verlag *Lecture Notes in Computer Science.*

Mechanical Verification of a Generalized Protocol for Byzantine Fault Tolerant Clock Synchronization

Natarajan Shankar
Computer Science Laboratory
SRI International
Menlo Park, CA 94025 USA

Schneider[1] generalizes a number of protocols for Byzantine fault-tolerant clock synchronization and presents a uniform proof for their correctness. We present a mechanical verification of Schneider's protocol leading to several significant clarifications and revisions. The verification was carried out with the EHDM system[2] developed at the SRI Computer Science Laboratory. The mechanically checked proofs include the verification that the *egocentric mean* function used in Lamport and Melliar-Smith's Interactive Convergence Algorithm[3] satisfies the requirements of Schneider's protocol. Our mechanical verification raises a number of issues regarding the verification of fault-tolerant, distributed, real-time protocols that are germane to the design of a special-purpose logic for such problems.

[1] Fred B. Schneider. Understanding protocols for Byzantine clock synchronization. Technical Report 87-859, Department of Computer Science, Cornell University, Ithaca, NY, August 1987.

[2] John Rushby, Friedrich von Henke, and Sam Owre. An introduction to formal specification and verification using EHDM. Technical Report SRI-CSL-91-2, Computer Science Laboratory, SRI International, Menlo Park, CA, February 1991.

[3] L. Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(1):52–78, January 1985.

37 File ftrtft94

From the Third International School and Symposium on "Formal Techniques in Real Time and Fault Tolerant Systems," 19–23 September 1994, Lübeck, Germany. Springer-Verlag Lecture Notes in Computer Science Vol. 863, pp. 660–679

Towards a Duration Calculus Proof Assistant in PVS

Jens Ulrik Skakkebæk
Department of Computer Science
Technical Univ. of Denmark
DK-2800 Lyngby, Denmark
Email: jus@id.dth.dk

Natarajan Shankar
Computer Science Laboratory
SRI International
Menlo Park CA 94025, USA
Email: shankar@csl.sri.com

The Duration Calculus (DC) is an interval temporal logic for reasoning about real-time systems. This paper presents a mechanized DC *proof assistant* which is a tool for constructing DC specifications and checking DC proofs. The proof assistant is implemented by encoding the semantics of DC within the higher-order logic of a general-purpose specification and verification environment called PVS. We have developed a Gentzen style sequent proof system for DC which we have proven to be sound with respect to the semantic encoding. We exploit the similarity between the sequent calculus of PVS and the sequent calculus of DC to obtain a DC proof system where the proofs are actually being carried out in PVS at the semantic level, but appear as proofs in the DC proof system. The result is a system called PC/DC that retains the specification and verification capabilities of PVS within the framework of the Duration Calculus. Thus, proofs constructed in PC/DC closely follow the lines of the corresponding pencil-and-paper DC arguments. We argue that our approach can be applied to specification logics other than DC. The DC sequent calculus, the semantic encoding of DC in PVS, and the DC proof checking tool are illustrated using the example of a simple gas burner.

38 File holwkshop91

Reprint of an invited paper from: Phillip Windley, editor, *Proceedings of the 1991 International Workshop on the HOL Theorem Proving System and its Applications*, Davis, CA, Aug. 1991, pages 195–204; IEEE Computer Society.

Design Choices in Specification Languages and Verification Systems

John Rushby

Computer Science Laboratory
SRI International
Menlo Park, CA 94025 USA

We describe some of the design choices that should be considered in the development and application of specification languages and verification systems. A principal issue is the need to reconcile the desire for expressiveness in the specification language with the ability to provide effective mechanical support. We argue that this reconciliation is assisted by a novel approach to specification language design that requires theorem proving to be used during typechecking.

A second key requirement is for the theorem prover to be specialized towards the needs of verification. This means that the theorem prover must assist in the rapid identification of the sources of errors in incorrect theorems as well as in the certification of true theorems, and that it must produce a proof suitable for human review. We argue that a combination of powerful automation (including decision procedures) at the low level, and user-guidance at the high level, provides the most effective approach to these goals.

39 File holwkshop93

Reprint from *Higher Order Logic Theorem Proving and its Applications (6th International Workshop, HUG '93)*, Vancouver, Canada, August, 1993. Springer-Verlag LNCS 780, pp. 163–173.

Using PVS to Prove Some Theorems of David Parnas

John Rushby and Mandayam Srivas
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

David Parnas[1] describes some theorems representative of those encountered in support of certification of software for the Darlington nuclear reactor. We describe the verification of these theorems using PVS.

Note a copy of Parnas' paper is available in this ftp directory as an uncompressed postscript file `proofs-wanted.ps`.

[1]David Lorge Parnas. Some theorems we should prove. In *Higher Order Logic Theorem Proving and its Applications (6th International Workshop, HUG '93)*, Vancouver, Canada, August, 1993. Springer-Verlag LNCS 780, pp. 155–162.

40 File informations91

Appeared in French Journal "T.A.Informations" 1991

LIFE

A Natural Language for Natural Language

Hassan Aït-Kaci and Patrick Lincoln
MCC, ACA Program
Systems Technology Laboratory
3500 West Balcones Center Drive
Austin, TX 78759

Experimenting with formalisms for Natural Language Processing involves costly programming overhead in conventional computing idioms, even as "advanced" as Lisp or Prolog. LIFE is a programming language which incorporates an elegant type system which supports a powerful facility for structured type inheritance. Also, LIFE reconciles styles from Functional Programming and Logic Programming by implicitly delegating control to an automatic suspension mechanism. This allows interleaving interpretation of relational and functional expressions which specify abstract structural dependencies on objects. Together, these features provide a convenient and versatile power of abstraction for very high-level expression of constrained data structures. Computational linguistics is a discipline where such abstractions are particularly useful. Therefore, obvious convenience is offered by LIFE for experimentation to the computational linguist, who becomes relieved from burdensome yet extrinsic programming complications. We presently attempt to show how LIFE may be a natural computer language for processing natural human languages.

41 File jsc89

Appeared in JSC 1989, vol 8

Adventures in Associative-Commutative Unification

Patrick Lincoln and Jim Christian
Department of Computer Science, Stanford University, Stanford, CA, 94305, USA
and
Department of Computer Science, University of Texas at Austin, Austin, TX, 78712, USA

We have discovered an efficient algorithm for matching and unification in associative-commutative (AC) equational theories. In most cases of AC unification our method obviates the need for solving diophantine equations, and thus avoids one of the bottlenecks of other associative-commutative unification techniques. The algorithm efficiently utilizes powerful constraints to eliminate much of the search involved in generating valid substitutions. Moreover, it is able to generate solutions lazily, enabling its use in an SLD-resolution-based environment like Prolog. We have found the method to run much faster and use less space than other associative-commutative unification procedures on many commonly encountered AC problems.

42 File lics91

Appeared in LICS '91

Linearizing Intuitionistic Implication (Extended Abstract)

Patrick Lincoln
Department of Computer Science, Stanford University, Stanford, CA 94305, and the Computer Science Laboratory, SRI International, Menlo Park, CA 94025.

Andre Scedrov
Department of Mathematics, University of Pennsylvania, Philadelphia, PA 19104-6395. Work begun while on sabbatical leave at the Computer Science Department and Center for the Study of Language and Information, Stanford University.

Natarajan Shankar
Computer Science Laboratory, SRI International, Menlo Park, CA 94025.

An embedding of the implicational propositional intuitionistic logic (IIL) into the nonmodal fragment of intuitionistic linear logic (IMALL) is given. The embedding preserves cut-free proofs in a proof system that is a variant of IIL. The embedding is efficient and provides an alternative proof of the PSPACE-hardness of IMALL. It exploits several proof-theoretic properties of intuitionistic implication that analyze the use of resources in IIL proofs.

43 File lics92

Appeared in ACM Symposium on Logic In Computer Science 1992

Operational Aspects of Linear Lambda Calculus

Patrick Lincoln
Computer Science Department
Stanford University
Stanford, CA, 94305
lincoln@cs.stanford.edu

John Mitchell
Computer Science Department
Stanford University
Stanford, CA, 94305
jcm@cs.stanford.edu

Linear logic is a resource-aware logic that is based on an analysis of the classical proof rules of contraction (copying) and weakening (throwing away). Several previous researchers have studied functional programming languages derived from linear logic according to the "formulas-as-types" correspondence. In languages with linear logic types, one may hope that traditional implementation problems in functional languages such as update in place could be simplified by careful use of the type system. In this paper, we prove that the standard sequent calculus proof system of linear logic is equivalent to a natural deduction style proof system. Using the natural deduction system, we investigate the pragmatic problems of type inference and type safety for a linear lambda calculus. Although terms do not have a single most-general type (for either the standard sequent presentation or our natural deduction formulation), there is a set of most-general types that may be computed using unification. The natural deduction system also facilitates the proof that the type of an expression is preserved by any evaluation step. We also describe an execution model and implementation, using a variant of the "three-instruction machine" (TIM). A novel feature of the implementation is that we distinguish garbage-collected "non-linear" memory from "linear" memory, which does not require garbage collection and where it is possible to do secure update in place.

44 File lics94

Reprint from: Ninth Annual IEEE Symposium on Logic in Computer Science (LICS'94) to be held in Paris, France, July 4-8 1994. Also appears in SRI Technical Report CSL-93-11.

Proof Search in First-order Linear Logic and Other Cut-free Sequent Calculi

P. D. Lincoln and N. Shankar
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

We present a general framework for proof search in first-order cut-free sequent calculi and apply it to the specific case of linear logic. In this framework, Herbrand functions are used to encode universal quantification, and unification is used to instantiate existential quantifiers so that the eigenvariable conditions are respected. We present an optimization of this procedure that exploits the permutabilities of the subject logic. We prove the soundness and completeness of several related proof search procedures. This proof search framework is used to show that provability for first-order MALL is in NEXPTIME, and first-order MLL is in NP. Performance comparisons based on Prolog implementations of the procedures are also given. The optimization of the quantifier steps in proof search can be combined effectively with a number of other optimizations that are also based on permutability.

45 File nasa-cr4596

Model-Based Reconfiguration: Diagnosis and Recovery

Judy Crow and John Rushby

Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025 USA

NASA Contractor Report 4596, May 1994

We extend Reiter's general theory of model-based diagnosis to a theory of fault detection, identification, and reconfiguration (FDIR). The generality of Reiter's theory readily supports an extension in which the problem of reconfiguration is viewed as a close analogue of the problem of diagnosis. Using a reconfiguration predicate **rcfg** analogous to the abnormality predicate **ab**, we derive a strategy for reconfiguration by transforming the corresponding strategy for diagnosis. There are two obvious benefits of this approach: first, algorithms for diagnosis can be exploited as algorithms for reconfiguration; second, we have a theoretical framework for an integrated approach to FDIR. As a first step toward realizing these benefits, we show that a class of diagnosis engines can be used for reconfiguration and we discuss algorithms for integrated FDIR. We argue that integrating recovery and diagnosis is an essential next step if this technology is to be useful for practical applications.

46 File nonint-verif

Formal Verification of the Unwinding Theorem for Noninterference Security Policies

John Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Draft Technical Report
April 1991

The concept of noninterference was introduced by Goguen and Meseguer[1] in order to provide a formal foundation for the specification and analysis of security policies and the mechanisms that enforce them. An important step in the application of noninterference formulations was the development of the unwinding theorem, which provides conditions on the behavior of individual operations that are sufficient to ensure secure behavior[2].

In this report we describe the formal specification and verification of an unwinding theorem for noninterference security policies using the EHDM formal specification and verification system[3]. The specification and verification of this theorem illustrates the use of EHDM in a simple but interesting setting, and provides an introduction to the rather more difficult verification of our version of the "multidomain security" or "intransitive" unwinding theorem[4,5].

This is a draft report circulated for comment and information. It will be incorporated into a set of EHDM examples to appear during 1993.

[1] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proceedings of the Symposium on Security and Privacy*, pages 11–20, Oakland, CA, April 1982. IEEE Computer Society.

[2] J. A. Goguen and J. Meseguer. Inference control and unwinding. In *Proceedings of the Symposium on Security and Privacy*, pages 75–86, Oakland, CA, April 1984. IEEE Computer Society.

[3] John Rushby, Friedrich von Henke, and Sam Owre. An introduction to formal specification and verification using EHDM. Technical Report SRI-CSL-91-2, Computer Science Laboratory, SRI International, Menlo Park, CA, February 1991.

[4] J. Thomas Haigh and William D. Young. Extending the noninterference version of MLS for SAT. *IEEE Transactions on Software Engineering*, SE-13(2):141–150, February 1987.

[5] John Rushby. Noninterference, transitivity, and channel-control security policies. Technical Report SRI-CSL-92-02, Computer Science Laboratory, SRI International, Menlo Park, CA, October 1992.

47 File podc94

From the 13th ACM Symposium on Principles of Distributed Computing (PODC '94), pp. 304–313, Los Angeles CA, August 1994

A Formally Verified Algorithm for Clock Synchronization Under a Hybrid Fault Model

John Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

A simple modification to the Interactive Convergence clock synchronization algorithm allows it to tolerate a larger number of certain kinds of faults than the standard algorithm, without reducing its tolerance to arbitrary or Byzantine faults. Because the extended case-analysis required by the new fault model complicates the already intricate argument for correctness of the algorithm, it has been subjected to mechanically-checked formal verification.

The fault model examined is similar to the "hybrid" one previously used for the problem of Interactive Consistency (Byzantine Agreement): in addition to *arbitrary* faults, we also admit *manifest* (i.e., detectable) and *symmetric* (i.e., consistent) faults. With n channels, the modified algorithm can withstand a arbitrary, s symmetric, and m manifest faults simultaneously, provided $n > 3a+2s+m$. This is better than the bound achieved for the problem of interactive consistency under the similar fault model. Unlike interactive consistency, we have been able to further extend the fault model for clock synchronization to include *link* faults with bound $n > 3a+2s+m+l$ where l is the maximum, over all pairs of channels, of the number of channels that have faulty links to one or other of the pair.

The mechanically-checked formal verification of the modified algorithm was achieved by extending one for the classical Interactive Convergence algorithm, and was accomplished in just a few days of work. A mechanically-checked formal specification and verification is a reusable intellectual resource whose initial cost is amply repaid by the support it provides for inexpensive and reliable investigation of modified assumptions and algorithms such as those reported here.

48 File popl92

Appeared in Proc ACM Symp. Principles of Programming Languages, 1992.

Algorithmic aspects of type inference with subtypes

Patrick Lincoln and John C. Mitchell
Computer Science Department,
Stanford University, Stanford, CA 94305

We study the complexity of type inference for programming languages with subtypes. There are three language variations that effect the problem: (i) basic functions may have polymorphic or more limited types, (ii) the subtype hierarchy may be fixed or vary as a result of subtype declarations within a program, and (iii) the subtype hierarchy may be an arbitrary partial order or may have a more restricted form, such as a tree or lattice. The naive algorithm for inferring a most general polymorphic type, under variable subtype hypotheses, requires deterministic exponential time. If we fix the subtype ordering, this upper bound grows to nondeterministic exponential time. We show that it is NP-hard to decide whether a lambda term has a type with respect to a fixed subtype hierarchy (involving only atomic type names). This lower bound applies to monomorphic or polymorphic languages. We give PSPACE upper bounds for deciding polymorphic typability if the subtype hierarchy has a lattice structure or the subtype hierarchy varies arbitrarily. We also give a polynomial time algorithm for the limited case where there are of no function constants and the type hierarchy is either variable or any fixed lattice.

49 File pvs-language

The PVS Specification Language (Beta Release)

S. Owre, N. Shankar and J. M. Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025
Phone: (415) 859-5272
{Owre,Shankar,Rushby}@csl.sri.com

The manual for the PVS specification language.

See also `pvs-prover`, `pvs-system`, `pvs-tutorial`, and `pvs-refcard`. The PVS system is available in directory `/pub/pvs`.

50 File pvs-prover

The PVS Proof Checker: A Reference Manual (Beta Release)

N. Shankar, S. Owre and J. M. Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025
Phone: (415) 859-5272
{Shankar,Owre,Rushby}@csl.sri.com

The manual for the PVS proof checker.

See also `pvs-language`, `pvs-system`, `pvs-tutorial`, and `pvs-refcard`. The PVS system is available in directory `/pub/pvs`.

51 File pvs-refcard

Reference Card for the PVS Specification and Verification System (Beta Release)

S. Owre and N. Shankar and J. M. Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025
{Owre,Shankar,Rushby}@csl.sri.com

A concise (8 page) reference to the PVS system, specification language, and prover.

See also `pvs-language`, `pvs-prover`, `pvs-system`, and `pvs-tutorial`. The PVS system is available in directory `/pub/pvs`.

52 File pvs-system

User Guide for the PVS Specification and Verification System (Beta Release)

S. Owre and N. Shankar and J. M. Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025
{Owre,Shankar,Rushby}@csl.sri.com

The manual for the PVS verification system.

See also `pvs-language`, `pvs-prover`, `pvs-tutorial`, and `pvs-refcard`. The PVS system is available in directory `/pub/pvs`.

53 File pvs-tutorial

Reprint from Tutorial Notes, *Formal Methods Europe '93: Industrial-Strength Formal Methods*, pages 357–406, Odense, Denmark, April 1993.

A Tutorial on Specification and Verification Using PVS

N. Shankar and S. Owre and J. M. Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

PVS stands for "Prototype Verification System." It consists of a specification language integrated with support tools and a theorem prover. PVS tries to provide the mechanization needed to apply formal methods both rigorously and productively.

This tutorial serves to introduce PVS. In the first section, we briefly sketch the purposes for which PVS is intended and the rationale behind its design, mention some of the uses that we and others are making of it, and explain how to get a copy of the system. In Section 2, we use a simple example to briefly introduce the major functions of PVS; Sections 3 and 4 then give more detail on the PVS language and theorem prover, respectively, also using examples. The PVS language, system, and theorem prover each have their own reference manuals, [1,2,3] which you will need to study in order to make productive use of the system. A pocket reference card, summarizing all the features of the PVS language, system, and prover is also available.

This tutorial does not introduce the general ideas of formal methods, nor explain how formal specification and verification can best be applied to various problem domains; rather, its purpose is to introduce some of the more unusual and powerful capabilities that are provided by PVS. Consequently, this document, and the examples we use, are somewhat technical and are most suitable for those who already have some experience with formal methods and wish to understand how PVS provides mechanized support for some of the more challenging aspects of formal methods.

See also `pvs-language`, `pvs-prover`, `pvs-system`, and `pvs-refcard`. The PVS system is available in directory `/pub/pvs`.

[1] S. Owre, N. Shankar, and J. M. Rushby. *The PVS Specification Language (Beta Release)*. Computer Science Laboratory, SRI International, Menlo Park, CA, February 1993.

[2] N. Shankar, S. Owre, and J. M. Rushby. *The PVS Proof Checker: A Reference Manual (Beta Release)*. Computer Science Laboratory, SRI International, Menlo Park, CA, February 1993.

[3] S. Owre, N. Shankar, and J. M. Rushby. *User Guide for the PVS Specification and Verification System (Beta Release)*. Computer Science Laboratory, SRI International, Menlo Park, CA, February 1993.

54 File repsa92

Reprint of an invited paper presented at an International Symposium (organized by CNES, the French Space Agency) on Real-Time Embedded Processing for Space Applications, Les Saintes-Maries-de-la-Mer, France, 3–6 November, 1992, pages 355–366. Published by Cépaduès-Éditions, Toulouse, France

Formal Methods for Dependable Real-Time Systems

John Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

We outline the motivation for using formal methods to specify and reason about real-time properties, and sketch some of the approaches that have been proposed and used.

55 File sigact92

Appeared in ACM SIGACT NEWS, Volume 23, Number 2 (Whole Number 83), Spring 1992, pp. 29–37.

Linear Logic

Patrick Lincoln
Computer Science Department,
Stanford University, Stanford, CA 94305

Linear logic was introduced by Girard in 1987. Since then many results have supported Girard's claims such as "Linear logic is a resource conscious logic". Increasingly, computer scientists have recognized linear logic as an expressive and powerful logic, with deep connections to concepts from computer science. The expressive power of linear logic is evidenced by some very natural encodings of computational models such as Petri nets, counter machines, Turing machines, and others.

This note presents an intuitive overview of linear logic, some recent theoretical results, and some interesting applications of linear logic to computer science.

56 File sigsoft91—Superseded by tse95

Reprint from *SIGSOFT '91: Software for Critical Systems*, December 4–6, New Orleans, LA. Published as ACM SIGSOFT Engineering Notes, Volume 16, Number 5, pp. 1–15

Formal Verification of Algorithms for Critical Systems

John Rushby and Friedrich von Henke[1]

Computer Science Laboratory
SRI International
Menlo Park, CA 94025, USA

We describe our experience with formal, machine-checked verification of algorithms for critical applications, concentrating on a Byzantine fault-tolerant algorithm for synchronizing the clocks in the replicated computers of a digital flight control system.

First, we explain the problems encountered in unsynchronized systems and the necessity, and criticality, of fault-tolerant synchronization. We give an overview of one such algorithm, and of the arguments for its correctness.

Next, we describe a verification of the algorithm that we performed using our EHDM system for formal specification and verification. We indicate the errors we found in the published analysis of the algorithm, and other benefits that we derived from the verification.

Based on our experience, we derive some key requirements for a formal specification and verification system adequate to the task of verifying algorithms of the type considered.

Finally, we summarize our conclusions regarding the benefits of formal verification in this domain, and the capabilities required of verification systems in order to realize those benefits.

Note: Superseded by tse95

[1]Current main affiliation: Fakultät für Informatik, Universität Ulm, Germany

57 File slp88

Appeared in IEEE Symposium on Logic Programming 1987

Le Fun:

Logic, equations, and Functions

Hassan Aït-Kaci, Patrick Lincoln and Roger Nasr
MCC, ACA Program
Systems Technology Laboratory
3500 West Balcones Center Drive
Austin, TX 78759

We introduce a new paradigm for the integration of functional and logic programming. Unlike most current research, our approach is not based on extending unification to general-purpose equation solving. Rather, we propose a computation delaying mechanism called *residuation*. This allows a clear distinction between functional *evaluation* and logical *deduction*. The former is based on the λ-calculus, and the latter on Horn clause resolution. In clear contrast with equation-solving approaches, our model supports higher-order function evaluation and efficient compilation of both functional and logic programming expressions, without being plagued by non-deterministic term-rewriting. In addition, residuation lends itself naturally to process synchronization and constrained search. Besides unification (equations), other residuations may be any ground-decidable goal, such as mutual exclusion (inequations), and comparisons (inequalities). We describe an implementation of the residuation paradigm as a prototype language called Le Fun—**L**ogic, **e**quations, and **Fun**ctions.

58 File toplas89

Appeared in TOPLAS 11-1-1989

Efficient Implementation of Lattice Operations

Hassan Aït-Kaci, Robert Boyer, Patrick Lincoln and Roger Nasr
MCC, ACA Program
Systems Technology Laboratory
3500 West Balcones Center Drive
Austin, TX 78759

Lattice operations such as greatest lower bound (GLB), least upper bound (LUB), and relative complementation (BUTNOT) are becoming more and more important in programming languages supporting object inheritance. We present a general technique for the efficient implementation of such operations based on an encoding method. The effect of the encoding is to plunge the given ordering into a boolean lattice of binary words, leading to an almost constant time complexity of the lattice operations. A first method is described based on a transitive closure approach. Then, a more space-efficient method minimizing code-word length is described. Finally, a powerful grouping technique called *modulation* is presented which drastically reduces code space while keeping all three lattice operations highly efficient. This technique takes into account idiosyncrasies of the topology of the poset being encoded which are quite likely to occur in practice. All methods are formally justified. We see this work as an original contribution towards using semantic (*vz.*, in this case, taxonomic) information in the engineering pragmatics of storage and retrieval of (*vz.*, partially or quasi-ordered) information.

59 File tpcd94 (available in postscript only)

Revised version of paper presented at the 2nd International Conference on Theorem Provers in Circuit Design, Bad Herrenalb (Blackforest), Germany, 26–29 September 1994, Ramayya Kumar and Thomas Kropf Eds., Forschungszentrum Informatik an der Universität Karlsruhe, FZI Publication 4/94, pp. 287–305

Effective Theorem Proving for Hardware Verification

D. Cyrluk, S. Rajan[1], N. Shankar, and M.K. Srivas

Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA
{cyrluk, sree, shankar, srivas}@csl.sri.com

The attractiveness of using theorem provers for system design verification lies in their generality. The major practical challenge confronting theorem proving technology is in combining this generality with an acceptable degree of automation. We describe an approach for enhancing the effectiveness of theorem provers for hardware verification through the use of efficient automatic procedures for rewriting, arithmetic and equality reasoning, and an off-the-shelf BDD-based propositional simplifier. These automatic procedures can be combined into general-purpose proof strategies that can efficiently automate a number of proofs including those of hardware correctness. The inference procedures and proof strategies have been implemented in the PVS verification system. They are applied to several examples including an N-bit adder, the Saxe pipelined processor, and the benchmark Tamarack microprocessor design. These examples illustrate the basic design philosophy underlying PVS where powerful and efficient low-level inferences are employed within high-level user-defined proof strategies. This approach is contrasted with approaches based on tactics or batch-oriented theorem proving.

[1]Integrated Systems Design Laboratory, Department of Computer Science, University of British Columbia.

60 File tpcd94-tutorial (available in postscript only)

Revised version of paper presented at the 2nd International Conference on Theorem Provers in Circuit Design, Bad Herrenalb (Blackforest), Germany, 26–29 September 1994, Ramayya Kumar and Thomas Kropf Eds., Forschungszentrum Informatik an der Universität Karlsruhe, FZI Publication 4/94, pp. 167–188

A Tutorial on Using PVS for Hardware Verification

S. Owre, J. M. Rushby, N. Shankar and M. K. Srivas
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

PVS stands for "Prototype Verification System." It consists of a specification language integrated with support tools and a theorem prover. PVS tries to provide the mechanization needed to apply formal methods both rigorously and productively.

This tutorial serves to introduce PVS and its use in the context of hardware verification. In the first section, we briefly sketch the purposes for which PVS is intended and the rationale behind its design, mention some of the uses that we and others are making of it. We give an overview of the PVS specification language and proof checker. The PVS language, system, and theorem prover each have their own reference manuals which you will need to study in order to make productive use of the system. A pocket reference card, summarizing all the features of the PVS language, system, and prover is also available.

The purpose of this tutorial is not to describe in detail the features of PVS and how to use the system. Rather, its purpose is to introduce some of the more unique and powerful capabilities that are provided by PVS and demonstrate how these features can be used in the context of hardware verification. We present completely worked out proofs of two hardware examples. One of the examples is a pipelined microprocessor that has been used as benchmark for model checkers and the other is a parameterized implementation of an N-bit ripple-carry adder.

61 File tse93

Reprint of paper in IEEE Transactions on Software Engineering, Volume 19, Number 1, pp 13–23, January 1993; A preliminary version was presented at *SIGSOFT '91: Software for Critical Systems*, December 4–6, New Orleans, LA (proceedings published as ACM SIGSOFT Engineering Notes, Volume 16, Number 5, pp. 1–15).

Formal Verification of Algorithms for Critical Systems

John Rushby and Friedrich von Henke

Computer Science Laboratory
SRI International
Menlo Park, CA 94025, USA

We describe our experience with formal, machine-checked verification of algorithms for critical applications, concentrating on a Byzantine fault-tolerant algorithm for synchronizing the clocks in the replicated computers of a digital flight control system.

First, we explain the problems encountered in unsynchronized systems and the necessity, and criticality, of fault-tolerant synchronization. We give an overview of one such algorithm, and of the arguments for its correctness.

Next, we describe a verification of the algorithm that we performed using our EHDM system for formal specification and verification. We indicate the errors we found in the published analysis of the algorithm, and other benefits that we derived from the verification.

Based on our experience, we derive some key requirements for a formal specification and verification system adequate to the task of verifying algorithms of the type considered.

Finally, we summarize our conclusions regarding the benefits of formal verification in this domain, and the capabilities required of verification systems in order to realize those benefits.

Index terms: Formal specification, formal verification, clock synchronization, fault-tolerant systems.

62 File tse95

Reporint of paper in IEEE Transactions on Software Engineering, Volume 21, Number 2, pp. 107–125, February 1995; A preliminary version appeared under the title "Formal Verification for Fault-Tolerant Architectures: Some Lessons Learned," in *FME '93: Industrial-Strength Formal Methods*, pages 482–500, Odense, Denmark, April 1993. Volume 670 of Springer Verlag Lecture Notes in Computer Science

Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS

Sam Owre, John Rushby,
Natarajan Shankar, Friedrich von Henke[1]
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

PVS is the most recent in a series of verification systems developed at SRI. Its design was strongly influenced, and later refined, by our experiences in developing formal specifications and mechanically checked verifications for the fault-tolerant architecture, algorithms, and implementations of a model "reliable computing platform" (RCP) for life-critical digital flight-control applications, and by a collaborative project to formally verify the design of a commercial avionics processor called AAMP5.

Several of the formal specifications and verifications performed in support of RCP and AAMP5 are individually of considerable complexity and difficulty. But in order to contribute to the overall goal, it has often been necessary to modify completed verifications to accommodate changed assumptions or requirements, and people other than the original developer have often needed to understand, review, build on, modify, or extract part of an intricate verification.

In this paper, we outline the verifications performed, present the lessons learned, and describe some of the design decisions taken in PVS to better support these large, difficult, iterative, and collaborative verifications.

[1]von Henke's main affiliation is Fakultät für Informatik, Universität Ulm, Germany.