Response to FM91 Survey of Formal Methods: Nqthm and Pc-Nqthm

Matt Kaufmann (with many contributors)

Technical Report 75 March, 1992

Computational Logic Inc.
1717 W. 6th St. Suite 290
Austin, Texas 78703
(512) 322-9951

This work was supported in part at Computational Logic, Inc., by the Defense Advanced Research Projects Agency, ARPA Order 7406. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of Computational Logic, Inc., the Defense Advanced Research Projects Agency or the U.S. Government.

Introduction

This report contains contributions to the requested survey for FM91, ‘‘A survey of formal methods tools and techniques’’ and ‘‘A survey of applications.’’ Two tools are covered: (1) Nqthm, the Boyer-Moore Theorem Prover; and (2) Pc-Nqthm, an interactive enhancement of Nqthm. Some of the words are taken from corresponding responses for FM89, but changes have been made.

The parts (other than this introduction) are as follows. In the text below, these are all separated by headers of the form

===== <section name> =====

1. Response for tool: Nqthm
2. Response for tool: Pc-Nqthm
3. Applications using Nqthm or Pc-Nqthm: Part I
4. Applications using Nqthm or Pc-Nqthm: Part II

The report concludes with references.

Perhaps some explanation of my treatment of Applications is in order. There have been many applications of Nqthm and of Pc-Nqthm, a number of them significant. Therefore, in order to convey this breadth in a reasonable amount of space, I’ve chosen in (3) to list essentially all of the applications that are on-line at Computational Logic, Inc. (the home of Nqthm and Pc-Nqthm), but with only a brief explanation of each. (4) is the treatment of a few of these and other applications using the requested format, so that depth can be obtained in addition to the breadth afforded by the more comprehensive listing in (3).

Acknowledgements. Many people contributed to this effort. In particular, the final section consists entirely of contributions made by the creators of the particular applications, with only trivial editing by this author. J Moore supplied the annotated list of Nqthm applications in Section 3. Without Boyer and Moore’s theorem prover, none of this would be possible. And without the supportive environment at Computational Logic, Inc., none of this would be as easy.

===== Section 1. Response for tool: Nqthm =====

Name: The Boyer-Moore Theorem Prover (Nqthm)

Participants: Robert S. Boyer and J Strother Moore

Contact name and address:

J Strother Moore
Computational Logic Inc.
1717 W. 6th St., Suite 290
Austin, TX 78703

phone: (512) 322-9951
email: moore@cli.com
fax: (512) 322-0656

Level of effort: 21 years X 2 persons

Description: This is an automatic/interactive theorem prover for a first order logic resembling Pure Lisp. The logic is based on recursive function definition and inductively defined data types. The theorem prover consists of about 1 megabyte of Common Lisp and contains many heuristics for controlling rule-based rewriting and induction.

Accomplishments: Proofs of Goedel's Incompleteness theorem, Wilson's Theorem, Gauss' law of quadratic reciprocity, unsolvability of the halting problem, correctness of the Boyer-Moore fast string searching algorithm, invertibility of the RSA encryption algorithm, Piton assembler, Micro-Gypsy compiler, FM8502 microprocessor, FM9001 microprocessor, KIT operating system, Bitonic sort, several programs in Unity, asynchronous communications, biphase mark protocol, 8-bit parallel io Byzantine agreement processor, a formalization of the Motorola MC68020, a variety of MC68020 machine code programs, a mutual exclusion algorithm, and many other problems.

Published articles and reports.

A Computational Logic, Boyer & Moore, Academic Press, 1979.

A Computational Logic Handbook, Boyer & Moore, Academic Press, 1988.

-- and, various articles on particular applications

Start date: 1971

Completion date: 1992

Future developments: A new theorem prover based on applicative Common Lisp with significantly greater control over hints, theories and libraries, an expanded set of built-in data types, random-access arrays, and many other improvements. A prototype currently exists.

Strengths/weaknesses/suitability: Sound, extensible, interactive via incorporation of user-suggested but mechanically proved rules. Complicated; requires experience. Seems particularly suited for proofs about computer systems, but has been used successfully on a wide variety of applications.

External users: Hundreds of copies have been distributed over the net. Mainly university research groups. Some industry interest. Much government interest.

Applications: See ‘‘Accomplishments’’ above as well as the ‘‘applications’’ portion of the survey.

Availability: To get a copy follow these instructions:

1. ftp to Internet host cli.com.
   (cli.com currently has Internet number 192.31.85.1)
2. log in as ftp, password guest
3. get the file /pub/nqthm/README
4. read the file README and follow the directions it gives.

Inquiries concerning tapes may be sent to: Computational Logic, Inc.,
Suite 290, 1717 W. 6th St., Austin, Texas 78703.

Additional remarks: See also the corresponding response for Pc-Nqthm.

This development of Nqthm has been an ongoing effort for 21 years. It has received support from a wide variety of sponsors including Science Research Council of Great Britain (now Science and Engineering Research Council), Xerox, SRI International, National Science Foundation, Office of Naval Research, NASA, Air Force Office of Scientific Research, Digital Equipment Corporation, the University of Texas at Austin, the Venture Research Unit of British Petroleum, Ltd., IBM, Defense Advanced Projects Research Agency, National Computer Security Center, the Space and Naval Warfare Systems Command, and Computational Logic, Inc. Our primary support now comes from Defense Advanced Research Projects Agency, DARPA Order 7406. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of Computational Logic, Inc., the Defense Advanced Research Projects Agency or the U.S. Government.

===== Section 2. Response for tool: Pc-Nqthm =====

Name: Pc-Nqthm (An interactive ‘‘Proof-checker’’ enhancement of the Boyer-Moore Theorem Prover

Participants: Matt Kaufmann

Contact name and address:

Matt Kaufmann
Computational Logic Inc.
1717 W. Sixth St., Suite 290
Austin, TX 78703

phone: (512) 322-9951
email: kaufmann@cli.com
fax: (512) 322-0656

Level of effort: Perhaps (very roughly) 2 man-years, spread over 5 years.

Description: This ‘‘proof-checker’’ is loaded on top of the Boyer-Moore Theorem Prover; see the corresponding response for Nqthm. The user can give commands at a low level (such as deleting a hypothesis, diving to a subterm of the current term, expanding a function call, or applying a rewrite rule) or at a high level (such as invoking the Boyer-Moore Theorem Prover). Commands also exist for displaying useful information (such as printing the current hypotheses and conclusion, displaying the currently applicable rewrite rules, or showing the current abbreviations) and for controlling the progress of the proof (such as undoing a specified number of commands, changing goals, or disabling certain rewrite rules). A notion of ‘‘macro commands’’ lets the user create compound commands, roughly in the spirit of the tactics and tacticals of LCF and its descendents. An on-line help facility is provided, and a user’s manual exists.

As with a variety of proof-checking systems, this system is goal-directed: a proof is completed when the main goal and all subgoals have been proved. Upon completion of an interactive proof, the lemma with its proof may be stored as a Boyer-Moore ‘‘event’’ that can be added to the user’s current library of definitions and lemmas. This event can later be replayed in ‘‘batch mode’’. Partial proofs can also be stored.

Accomplishments: This system has been used to check theorems stating the correctness of a transitive closure program, a Towers of Hanoi program, a ground resolution prover, a compiler, irrationality of the square root of 2, an algorithm of Gries for finding the largest "true square" submatrix of a boolean matrix, the exponent two version of Ramsey’s Theorem, the Shroeder-Bernstein theorem, Koenig’s tree lemma, and others. It has also been used to check the correctness of several Unity programs and has been used for hardware verification.

Published articles and reports. The first one below is a detailed user’s manual, including soundness arguments. The second extends this by describing an extension of the system which admits free variables, an important addition for doing full first-order reasoning. The third is a reference for that full first-order reasoning capability.

Matt Kaufmann, A User’s Manual for an Interactive Enhancement to the Boyer-Moore Theorem Prover. Technical Report 19, Computational Logic, Inc., May, 1988.

Matt Kaufmann, Addition of Free Variables to an Interactive Enhancement of the Boyer-Moore Theorem

Prover. Technical Report 42, Computational Logic, Inc., May, 1989.

Matt Kaufmann, An Extension of the Boyer-Moore Theorem Prover to Support First-Order Quantification. To appear in J. of Automated Reasoning" A preliminary (and expanded) version appears as Technical Report 43, Computational Logic, Inc., May, 1989.

-- and, various articles on particular applications.

Start date: Late 1986

Completion date: 1992

Future developments: Boyer and Moore are creating a new theorem prover based on applicative Common Lisp; see the ‘‘Future developments’’ response for Nqthm. There currently exists a similar interactive enhancement for that system, and we intend to continue development of that capability.

Strengths/weaknesses/suitability:

Strengths include:

- Combination of capability for high degree of user control with the power of the Boyer-Moore prover
- On-line help facility and users manuals
- Extensibility by way of ‘‘macro commands’’ (patterned after the tactics and tacticals of LCF, HOL, Nuprl etc.)
- Full integration into Boyer-Moore system
- Careful attention to soundness issues

Weaknesses include:

- Ease of low-level interaction often tempts users to construct ugly proofs without many reusable lemmas that are hard to modify.
- First-order quantification is handled via Skolemization, rather than directly (as in the Never prover).

External users: Dozens of copies have been distributed over the net. Mainly university research groups. Some industry interest. Much government interest.

Applications: See ‘‘Accomplishments’’ above as well as the ‘‘applications’’ portion of the survey.

Availability: To get a copy, first obtain the Boyer-Moore Theorem Prover (Nqthm), as described in the corresponding response for Nqthm. Then follow these instructions:

1. ftp to Internet host cli.com.
   (cli.com currently has Internet number
   192.31.85.1)
2. log in as ftp, password guest
3. get the file /pub/proof-checker/README-pc
4. read the file README-pc and follow the directions it gives.

Inquiries concerning tapes may be sent to: Computational Logic, Inc.,
Suite 290, 1717 W. 6th St., Austin, Texas 78703.

Additional remarks: See also the corresponding response for Nqthm.

This work was supported by the Defense Advanced Research Projects Agency (currently DARPA Order 7406), the Office of Naval Research, the National Computer Security Center, and IBM. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of Computational Logic, Inc., the Defense Advanced Research Projects Agency or the U.S. Government.

===== Section 3. Applications survey, Nqthm and Pc-Nqthm: Part I =====

Below is a list all of the prover input files that we have on-line at Computational Logic, Inc. Many of these example files are freely distributed by their authors. In Part II we present detailed summaries of a few of the larger applications, in the format requested.

J Moore has supplied the information below in the case of Nqthm, and Matt Kaufmann has done so for Pc-Nqthm. In each case, we give a very rough indication of level of effort by stating the number of bytes for the input file.

For each entry we give a partial file name, followed by the number of bytes for that file. That file length is followed in parentheses by the author(s) and possibly a citation, followed by a very brief description of the work. When no citation is given, no published description of the work is available and the interested reader should look at the events file itself. Many of the files have explanatory comments.

The files are listed in alphabetical order for Nqthm, and again for Pc-Nqthm. NOTE!!! The sizes and complexity of these applications vary wildly. The brief descriptions and file lengths should give a little indication of the relative sizes of the efforts.

References appear at the end of the paper.

------------------- Nqthm applications -------------------

I have been told that there are about 16,000 theorems (PROVE-LEMMA forms) in the Nqthm event files discussed below.

`basic/alternating` (14237 bytes)
(Boyer) a formalization and correctness proof of a card trick having to do with the outcome of shuffling a deck of cards that has been previously arranged into alternating colors

`basic/async18` (150871 bytes)
(Moore, [1]) a model of asynchronous communication and a proof of the reliability of the biphase mark communications protocol

`basic/binomial` (4015 bytes)
(Boyer and Moore, [2]) the binomial theorem expressed with **`FOR`** and a proof thereof

`basic/controller` (9184 bytes)
(Boyer, Moore, and Green, [3]) a model of the problem of controlling a vehicle's course and a proof that under certain conditions a particular program keeps the vehicle within a certain corridor of the desired course and, under more ideal conditions, homes to the course

`basic/fibsums` (13243 bytes)
(Cowles) proofs of several interesting theorems about the sums of Fibonacci numbers

`basic/fortran` (10687 bytes)
(Boyer and Moore, [4]) supporting definitions for a Fortran verification condition generator

`basic/fs-examples` (9243 bytes)
(Boyer, Goldschlag, Kaufmann, and Moore, [5]) illustrations of the use of constrained functions and functional instantiation

`basic/gauss` (55601 bytes)
(Russinoff, [6]) the original Nqthm proof of Gauss' law of quadratic reciprocity

`basic/new-gauss` (37455 bytes)
(Russinoff, [6]) an improved proof of Gauss' law of quadratic reciprocity (after all, Gauss proved it eight times!)

`basic/parser` (Boyer and Moore, an appendix of [7]) a formalization of the syntax and abbreviation conventions of the Nqthm extended logic, expressed as a function from lists of ASCII character codes to the quotations of formal terms

`basic/peter` (8376 bytes)
(Boyer) a sequence of lemmas describing the relationship between Ackermann's original function and R. Peter's version of it

`basic/pr` (7863 bytes)
(Boyer) a proof of the existence of nonprimitive recursive functions

`basic/proveall` (86983 bytes)
(Boyer and Moore, Appendix A of [8]) elementary list processing, number theory through Euclid's theorem and prime factorization, soundness and completeness of a tautology checker, correctness of the **`CANCEL`** metafunction, correctness of a simple assembly language program, correctness of a simple optimizing expression compiler

`basic/quant` (59573 bytes)
(Boyer and Moore, [2]) illustrations of the use of **`V&C$`** and **`FOR`**, including a study of several partial functions and functions, such as the ''91 function'' that recurse on the value of their own recursive calls

`basic/rsa` (17481 bytes)
(Boyer and Moore, [9]) proof of the invertibility of the Rivest/Shamir/Adleman public key encryption algorithm

`basic/small-machine` (25494 bytes)
: (Moore) a simple operational semantics and its use to prove program properties directly and via the so-called ‘‘functional’’ and ‘‘inductive assertion’’ methods

`basic/tic-tac-toe` (77473 bytes)
: (Moore) a formalization of what it means for a program to play non-losing tic-tac-toe, the proof that a certain algorithm does so, and the successive refinement of the algorithm into the functional expression of an iterative number-crunching program

`basic/tmi` (17429 bytes)
: (Boyer and Moore, [10]) proof of the Turing completeness of Pure Lisp

`basic/unsolv` (13796 bytes)
: (Boyer and Moore, [11]) proof of the unsolvability of the halting problem for Pure Lisp

`basic/wilson` (17576 bytes)
: (Russinoff, [12]) proof of Wilson’s theorem

`basic/ztak` (7065 bytes)
: (Moore, [13]) proof of the termination of Takeuchi’s function

`bevier/kit` (3258803 bytes)
: (Bevier, [14] the formalization, implementation and proof that a simple separation kernel (implementing multi-processing on a uniprocessor) provides process scheduling, error handling, message passing, and interfaces to asynchronous devices

`bronstein/*` (360053 bytes)
: (Bronstein, [15, 16, 17] hardware verification using string-functional semantics) provides the set of events described in Alex Bronstein’s dissertation.

`cowles/intro-eg` (4902 bytes)
: (Cowles) a brief introduction to Nqthm intended for mathematicians and a proof of a theorem about factorial

`cowles/shell` (9703 bytes)
: (Cowles) alternative ways to decompose sequences and a study of Nqthm’s shell principle

`fm9001-piton/big-add` (31064 bytes)
: (Moore, [18]) a proof of the correctness of a Piton program for adding arbitrarily long numbers in base 2^{32}

`fm9001-piton/fm9001` (2175131 bytes)
: (Brock and Hunt, [19]) formalizations of a netlist description language, the machine code for the 32-bit FM9001 microprocessor, the design of an implementation of the processor, and a proof of the correspondence of the design and the machine code specification

`fm9001-piton/piton` (1709400 bytes)
: (Moore, [18]) the definition of the Piton assembly language, its implementation on the FM9001 via a compiler, assembler and linker, and a proof of the correctness of the FM9001 implementation

`fortran-vcg/fortran` (10686 bytes)
: (Boyer and Moore, [4]) the same file as **`basic/fortran`**, above, which is duplicated on this subdirectory for technical reasons

`fortran-vcg/fsrch` (59138 bytes)
: (Boyer and Moore, [4]) proofs of the verification conditions for a Fortran implementation of the Boyer-Moore fast string searching algorithm

`fortran-vcg/isqrt` (12526 bytes)
: (Boyer and Moore, [20]) proofs of the verification conditions for a Fortran implementation of the integer version of Newton’s square root algorithm

`fortran-vcg/mjrty` (62019 bytes)
: (Boyer and Moore, [21]) proofs of the verification conditions for a Fortran implemen-

tation of a linear-time majority vote algorithm

hunt/fm8501 (301605 bytes)
(Hunt, [22]) formalizations of the machine code for the 16-bit FM8501 microprocessor, a register transfer model of a microcoded implementation of the machine, and a proof of their correspondence

kaufmann/expr-compiler (4365 bytes)
(Kaufmann, see Young [23]) the proof of correctness of a simple expression compiler, designed as an exercise for beginners

kaufmann/foldr (10895 bytes)
(Kaufmann) an illustration of a method of proving permutation-independence of list processing functions

kaufmann/generalize-all (106112 bytes)
(Kaufmann, [24]) the correctness of a generalization algorithm that operates in the presence of free variables

kaufmann/koenig (15825 bytes)
(Kaufmann, [25]) a proof of Koenig's tree lemma

kaufmann/locking (6177 bytes)
(Kaufmann, [26]) a model of a simple data base against which read and write transactions can occur

kaufmann/mergesort-demo (4833 bytes)
(Kaufmann) the correctness of a merge sort function

kaufmann/note-100 (10071 bytes)
(Kaufmann, [27]) the proof of Ramsey's theorem for exponent 2, finite case, described in a style intended to assist those wishing to improve their effectiveness with Nqthm

kaufmann/partial (9925 bytes)
(Kaufmann) an approach to handling partial functions with Nqthm

kaufmann/permutationp-subbagp (2018 bytes)
(Kaufmann) a formalization of the notion of permutation via bags

kaufmann/ramsey (17010 bytes)
(Kaufmann, [25]) a proof of Ramsey's theorem for the infinite case

kaufmann/rotate (1900 bytes)
(Kaufmann, [28]) a proof about rotations of lists, intended as an introduction to Nqthm

kaufmann/rpn (3028 bytes)
(Kaufmann and Jamsek) an exercise in reverse Polish notation evaluation

kunen/ack (3520 bytes)
(Kunen) an illustrative definition of Ackermann's function

kunen/new-prime (39987 bytes)
(Kunen) an alternative proof of the fundamental theorem of arithmetic that -- unlike the one presented in [8] -- does not use concepts not involved in the statement of the theorem

numbers/bags (4888 bytes)
(Bevier) a library of useful definitions and lemmas about bags

numbers/extras (377 bytes)
(Wilding) a trivial extension of the **integers** library used in **fib2** below

numbers/fib2 (15871 bytes)
(Wilding, [29]) a proof of Matijasevich's lemma about Fibonacci numbers

numbers/integers (236757 bytes)
(Bevier, Kaufmann, and Wilding, [30]) a library of useful definitions and lemmas

about the integers

numbers/naturals (100476 bytes)
(Bevier, Kaufmann, and Wilding, [31]) a library of useful definitions and lemmas about the natural numbers

numbers/nim (51396 bytes)
(Wilding) a formalization of the game of Nim and a proof that a certain algorithm implements a winning strategy

shankar/church-rosser (61967 bytes)
(Shankar, [32]) a proof of the Church-Rosser theorem for lambda-calculus

shankar/goedel (1054073 bytes)
(Shankar, [33]) a proof of Goedel's incompleteness theorem for Shoenfield's first order logic extended with Cohen's axioms for hereditarily finite set theory, Z2

shankar/tautology (68347 bytes)
(Shankar, [34]) a proof that every tautology has a proof in Shoenfield's propositional logic

talcott/mutex-atomic (127131 bytes)
(Nagayama and Talcott, [35]) a proof of the local correctness of a mutual exclusion algorithm under a certain ‘‘atomicity assumption’’

talcott/mutex-molecular (169218 bytes)
(Nagayama and Talcott, [35]) a proof of the local correctness of a mutual exclusion algorithm without the ‘‘atomicity assumption’’ mentioned above

yu/bsearch (17548 bytes)
(Yu, [36]) the correctness proof for the MC68020 machine code produced by the Gnu C compiler for a binary search program written in C

yu/gcd (13392 bytes)
(Yu, [36]) the correctness proof for the MC68020 machine code produced by the Gnu C compiler for Euclid's greatest common divisor algorithm written in C

yu/group (31947 bytes)
(Yu, [37]) proofs of two theorems in finite group theory, the first is about kernels of homomorphisms and the second is the Lagrange theorem

yu/isqrt-ada (14118 bytes)
(Yu) the correctness proof for the MC68020 machine code produced by the Verdix Ada compiler from an Ada program for computing integer square roots via Newton's method written in C

yu/mc20-0 (26383 bytes)
(Yu, [36]) some utilities involved in the formal specification of the MC68020

yu/mc20-1 (165631 bytes)
(Yu, [38]) the formal specification of about 80% of the user available instructions for the Motorola MC68020 microprocessor

yu/mc20-2 (166450 bytes)
(Yu, [36]) a library of useful definitions and lemmas about the formalization of the MC68020

yu/mjrty (33507 bytes)
(Yu) the correctness proof for the MC68020 machine code produced by the Gnu C compiler for a linear time majority vote algorithm written in C

yu/qsort (65665 bytes)
(Yu) the correctness proof for the MC68020 machine code produced by the Gnu C compiler for Hoare's *in situ* quick sort program written in C

yu/log2 (9212 bytes)
(Yu) the correctness proof for the MC68020 machine code produced by the Gnu C compiler for a C program for computing integer logarithms (base 2) e.g., repeated

division by 2

yu/cstring, yu/mem*.events, yu/str*.events (251139 bytes)
(Yu) verifications of MC68020 machine code of 15 of the 19 C String Library functions from the Berkeley Unix C String Library

------------------- Pc-Nqthm applications --------------------

Disclaimer: Many of the events in "basic/" are inelegant, as they were worked out in the early phases of the development of Pc-Nqthm. They are not to be viewed as models of good usage!

Some of these use the DEFN-SK quantifier enhancement [25], which may or may not be included in a final release of Pc-Nqthm but will remain available in any case.

`basic/arith.events` (10632 bytes)
some supporting arithmetic events for other event files in this directory

`basic/hanoi.events` (15461 bytes)
(Kaufmann) proof of correctness of a Towers of Hanoi program

`basic/pigeon-hole.events` (2369 bytes)
(Kaufmann) proof of a version of the pigeon-hole principle

`basic/ramsey1.events` (12633 bytes)
(Kaufmann) Our original proof of correctness of Ramsey's Theorem for exponent 2

`basic/ramsey2.events` (1792 bytes)
(Kaufmann) proof that a certain binomial coefficient serves as a bound on the Ramsey number

`basic/square.events` (5798 bytes)
(Kaufmann) ugly proof of an ugly formalization of the irrationality of the square root of 2

`basic/subset.events` (5025 bytes)
(Kaufmann) some supporting events about lists and their use as an implementation of sets

`basic/symmetric-difference.events` (3221 bytes)
(Kaufmann) commutativity and associativity of symmetric difference as a set operation

`basic/transitive-closure.events` (17530 bytes)
(Kaufmann) proof of correctness of a transitive closure algorithm

`basic/tsquare.events` (40440 bytes)
(Kaufmann) proof of correctness of a ‘‘true square’’ algorithm of Gries

`defn-sk/csb.events` (18698 bytes)
(Kaufmann, [25]) proof of a formalization of the Schroeder-Bernstein theorem of set theory

`defn-sk/finite-state-machine-example.events` (8062 bytes)
(Kaufmann) a little finite state machine example

`defn-sk/koenig.events` (13469 bytes)
(Kaufmann, [25]) a formalization of Koenig's Tree Lemma, which says that any finitely branching tree which is infinite has an infinite branch

`defn-sk/ramsey.events` (13514 bytes)
(Kaufmann, [25]) proof of a formalization of the infinite Ramsey Theorem for exponent 2

`dmg/bags.events` (4480 bytes)
(Bevier, [31]) some supporting events about bags

`dmg/dining.events` (86876 bytes)
(Goldschlag [39]) the verification of a dining philosopher's program, under the assumptions of deadlock freedom and strong fairness, using a mechanized implementation of Unity on the Boyer-Moore prover

`dmg/fifo.events` (148159 bytes)
(Goldschlag [40]) the verification of both the safety and liveness properties of an

n-node delay insensitive fifo circuit, using a mechanized implementation of Unity on the Boyer-Moore prover

dmg/interpreter.events (59999 bytes)
(Goldschlag [41]) a mechanized implementation of Unity on the Boyer-Moore prover

dmg/me.events (36101 bytes)
(Goldschlag) verification of an n-processor program satisfying both mutual exclusion and absence of starvation, using a mechanized implementation of Unity on the Boyer-Moore prover

dmg/min.events (261122 bytes)
(Goldschlag) the correctness of a distributed algorithm that computes the minimum node value in a tree, using a mechanized implementation of Unity on the Boyer-Moore prover

dmg/naturals.events (77298 bytes)
(Bevier, Wilding [31]) some supporting events about natural numbers

generalize/*.events (92003 bytes)
(Kaufmann, [24]) the correctness of a generalization algorithm that operates in the presence of free variables; same as **kaufmann/generalize-all** from the Nqthm suite, except that the quantifier (DEFN-SK, [25]) events have been replaced by DCL and ADD-AXIOM events in that version

mg/*.events (1961835 bytes)
(Young, [42]) a mechanically-verified code-generator for micro-Gypsy, which is a Pascal-like language

middle-gypsy/*.events (2048662 bytes)
(Good, Siebert, Young, [43]) a mathematical definition of a subset of the Gypsy 2.05 language, including a preliminary rationals library created by Matt Wilding[1]

wilding/ground-resolution.events (80997 bytes)
(Wilding) a proof of the completeness of ground resolution using Bledsoe's excess literal technique

[1]This rationals library contains Pc-Nqthm ''Instructions'' hints. If not for that, it seems quite possible that all of this would replay in Nqthm, with the DEFN-SK enhancement loaded.

===== Section 4. Applications survey, Nqthm and Pc-Nqthm: Part II =====

Here we present detailed summaries of a few of the larger applications, in the format requested. These have been supplied by participants in the respective applications, with at most minor editing by Matt Kaufmann. These are listed alphabetically by the first last name in ‘‘participants’’ (got that?).

Name of application project: CLI Verified Stack

Participants: Bill Bevier, Warren Hunt, J Moore, Bill Young

Contact name and address:

Dr. Bill Young
Computational Logic, Inc.
1717 W. 6th St., Suite 290
Austin, TX 78703

(512) 322-9951
email: young@CLI.COM

Level of effort: 4 people, around 2 years

Brief description:

The CLI ‘‘verified stack’’ is a collection of system components, each built upon the previous one, each subsequently providing more abstraction, and each being formally specified and mechanically verified using the Boyer-Moore theorem prover ‘Nqthm’ (and in the case of Bill Young’s work, the interactive enhancement ‘Pc-Nqthm’ of Nqthm). The system we have constructed contains a microprocessor with machine code, an assembly language, and a simple high-level programming language, together with the compiler, assembler and linker necessary to connect them. The stack properly consists of four abstract machines:

1. Gates--a register-transfer model of a microcoded machine.
2. FM8502--a machine code interpreter. More recently, this has been replaced by FM9001, which is a hardwired, 32-bit machine.
3. Piton--a high-level assembly language.
4. Micro-Gypsy--a simple high-level language which is a subset of the Gypsy 2.05 programming and specification language.

Relating each pair of adjacent machines is an implementation, represented as a function in the Boyer-Moore logic, that maps a higher-level state into a lower-level state. The implementation function is known as a ‘‘compiler’’ for the step from Micro-Gypsy to Piton, but as a ‘‘link-assembler’’ for the step from Piton to FM8502.

In addition, we have specified and proved correct the implementation a small operating system kernel (KIT) written for a uniprocessor von Neumann machine. KIT is proved to implement a fixed number of conceptually distributed communicating processes on this shared computer. In addition to implementing processes, Kit provides the following verified services: process scheduling, error handling, message passing, and an interface to asynchronous devices. KIT is not currently integrated into our stack.

We believe that this is the first hierarchically verified system of such complexity. It is possible using these components to compose and verify a high level program, and generate a core image for a microprocessor which provably preserves the semantics. We believe that future progress in this direction will lead to a higher degree of reliability than any existing approach.

We found many errors during the development of these systems. The discovery of these errors was a direct result of the use of a theorem prover: failed proofs would point us to the errors in our definitions. Thus, we have relearned a lesson that we already knew: although it is expensive, mechanical verification can assist in the discovery of errors and thus contribute significantly to the development of high-reliability systems.

Published articles and reports:

Journal of Automated Reasoning, Vol. 5, No. 4 (November, 1989) contains 5 papers describing the stack and the individual components.

Name of application project: PREVAIL

Participants: D.Borrione, L.Pierre, A.Salem, C.Le Faou, S.Coupet

Contact name and address:

D.Borrione or L.Pierre
IMAG/ARTEMIS, BP 53X
38041 Grenoble Cedex, FRANCE
email : borrione@imag.fr or lolo@imag.fr

Level of effort : 2 person-years

Brief description :

- Outline the nature of the application, the tools/techniques employed:

PREVAIL is a prototype environment for the formal proof of digital devices. It takes as input VHDL circuit descriptions, and automatically verifies their correctness. Several proof tools are included in this system, and NQTHM is one of them.

- What role did the formal methods play in the development process?

We have defined the functional semantics of a subset of VHDL. Then, inspired by the works of W.Hunt as reported in his PhD thesis, we have written a translator which automatically produces the function definitions and the theorems to be proven, under the input format for NQTHM.

- What were the benefits?

We use this prover in order to perform some kinds of proofs that are impossible with other tools, such as boolean tautology checkers. The main advantages are the possibility of reasoning on parameterized devices, and its ability to deal simultaneously with bit-vectors and integers.

- What were the drawbacks?

The major drawback, from our point of view, is the difficulty to identify whether a proof failed because the device is erroneous or because the theorem is not written in the right way. In addition, it is almost impossible to return a meaningful diagnosis to the user when there is a fault.

- What lessons can be drawn, recommendations made?

In view of the previous remark, it would be useful to have a summarized history of the proof (with the essential points) as an alternative to the complete one.

Published articles and reports :

L.PIERRE : The Formal Proof of Sequential Circuits described in CASCADE using the Boyer-Moore Theorem Prover.'' Proc. IFIP WG 10.2 Int. Workshop Nov. 1989. In ‘‘Formal VLSI Correctness Verification,'' L.Claesen Ed., North Holland, 1990, ISBN 0444 88689 3.

L.PIERRE : ‘‘One Aspect of Mechanizing Formal Proof of Hardware : the Generalization of Partial Specifications.’’ Proc. ACM International Workshop on Formal Methods in VLSI Design. Miami (Florida). 9-11 January 1991.

L.PIERRE : ‘‘From a HDL Description to Formal Proof Systems : Principles and Mechanization.’’ Proc. 10th International Symposium on Computer Hardware Description Languages and their Applications. Marseille (France). 22-24 April 1991.

S.COUPET, L.PIERRE : ‘‘Recursive Models for Synchronous Sequential Devices.’’ Research report IMAG/ARTEMIS 855-I, Grenoble. July 1991.

C.LE FAOU, L.PIERRE, A.SALEM : ‘‘A user-oriented presentation of PREVAIL : a proof environment for VHDL descriptions.’’ Technical report IMAG/ARTEMIS RT71, Grenoble. September 1991.

D.BORRIONE, L.PIERRE, A.SALEM : ‘‘PREVAIL : a proof environment for VHDL descriptions.’’ Proc. Advanced Research Workshop on Correct Hardware Design Methodologies, Torino, Italy, June 12-14, 1991.

D.BORRIONE, L.PIERRE, A.SALEM : ‘‘Formal Verification of VHDL Descriptions in the PREVAIL Environment.’’ To appear in a Special Issue of IEEE Design&Test on VHDL, 1992.

Name of application project: Compiler for NQTHM Logic

Participants: Arthur Flatau

Contact name and address:

Arthur Flatau
Computational Logic Inc.
1717 W. 6th St., Suite 290
Austin, TX 78703

(512) 322-9951
email flatau@cli.com

Level of effort: approx 3 man-years

Brief description:

A compiler from the NQTHM logic to the Piton assembly level language is being built. The compiler will include a reference count garbage collector. The compiler is written and formally specified in the NQTHM logic.

Currently a prototype and its proof of correctness has been completed and work is underway to add the garbage collector.

Published articles and reports:

Arthur Flatau, ‘‘A Compiler for NQTHM: A Progress Report.’’ Technical Report 74, Computational Logic, Inc., February 1992.

Name of application project: Mechanically Verifying Concurrent Programs

Participants: David Goldschlag

Contact name and address:

David Goldschlag
National Security Agency
Attn: R232
Fort George G. Meade, Maryland
20755-6000
Voice: (410) 859-4494
Fax: (410) 850-7166
email: goldschlag@tycho.ncsc.mil

Level of effort: Four person years over four years

Brief description:

This project embeds Chandy and Misra's Unity logic on Kaufmann's proof checker extension of the Boyer-Moore prover, and demonstrates this theory by the mechanical verification of four simple, yet parameterized, concurrent programs:

- a dining philosopher's program proved under the assumptions of strong fairness and absence of deadlock;
- an n-node delay insensitive fifo circuit;
- an n-processor program satisfying both mutual exclusion and absence of starvation; and
- a distributed algorithm that computes the minimum node value in a tree.

Various first order safety and liveness properties are proved. These mechanized proofs are expensive, however.

This mechanized proof system is novel, since it is sound by construction. Unity's proof rules are proved as theorems about an arbitrary fair computation, rather than presented as axioms characterizing a logic.

This methodology of building a proof system on top of an operational semantics is a reasonable and useful technique for developing sound proof systems.

Published articles and reports

D.M. Goldschlag, A Mechanical Formalization of Several Fairness Notions, in VDM '91: Formal Software Development Methods, S. Prehn and W.J. Toetenel (Editors), Springer-Verlag Lecture Notes in Computer Science 551, Springer-Verlag, Berlin, 1991.

D.M. Goldschlag, Mechanically Verifying Safety and Liveness Properties of Delay Insensitive Circuits, Computer Aided Verification 1991, Aalborg, Denmark, July 1991.

D.M. Goldschlag, Verifying Safety and Liveness Properties of a Delay Insensitive FIFO Circuit on the Boyer-Moore Prover, 1991 International Workshop on Formal Methods in VLSI Design, Miami, January, 1991.

D.M. Goldschlag, Mechanically Verifying Concurrent Programs with the Boyer-Moore Prover, IEEE Transactions on Software Engineering, September 1990.

D.M. Goldschlag, Proving Proof Rules: A Proof System for Concurrent Programs, Fifth Annual Conference on Computer Assurance, Compass 1990, Maryland, June 1990.

D.M. Goldschlag, Mechanizing Unity, in Programming Concepts and Methods, M. Broy and C.B. Jones (editors), North-Holland, Amsterdam, 1990.

J.M. Crawford and D.M. Goldschlag, Mechanical Verification, The Twenty-Sixth Annual Lake Arrowhead Workshop: How Will We Specify Concurrent Systems in the Year 2000?, Lake Arrowhead, California, September, 1987.

Name of application project: Generalization in the Presence of Free Variables: A Mechanically-Checked Proof for One Algorithm

Participants: Matt Kaufmann

Contact name and address:

Matt Kaufmann
Computational Logic Inc.
1717 W. 6th St., Suite 290
Austin, TX 78703

(512) 322-9951
email kaufmann@cli.com

Level of Effort: Roughly 2 man-months

Brief Description: Some of the following is adapted from the paper referenced below.

The motivation for this work began with a concern generated by a soundness bug in the Pc-Nqthm command GENERALIZE. This bug could be viewed as resulting from a delicate interaction between the first-order inference rule of universal instantiaion and ‘‘free’’ (instantiatable, Skolem) variables. At any rate, the bug was easily corrected and the correctness of the resulting GENERALIZE command was checked quite informally on paper (though that wasn’t quite as easy). However, the rude shock of having made a soundness mistake in the previous version of Pc-Nqthm led to the following goal: formalize the new version of the GENERALIZE command in the Boyer-Moore logic, and mechanically check a proof of correctness of this formalization.

This work consists of a mechanically-checked proof of correctness for a generalization algorithm. Although the theorem itself is probably new (at least, we are unaware of any existing statement of it), the interest here lies not particularly in the theorem *per se* but, rather, lies in the demonstration of the use of an automated reasoning assistant to check the reliability of detailed proofs and software. What is new about the current effort is the use of the Boyer-Moore prover (slightly extended) to check a non-trivial logic proof related to the correctness of an actual implementation, though it must be conceded that the actual Pc-Nqthm code is not a direct translation of the definitions from this proof, but is only conceptually related to them.

The paper referenced below is written with the intention of providing a good introduction to this style of mechanically-assisted reasoning.

Published articles and reports:

Matt Kaufmann, ‘‘Generalization in the Presence of Free Variables: A Mechanically-Checked Proof for One Algorithm.’’ J. of Automated Reasoning 7, March, 1991.

Name of application project: A Formal Model of Asynchronous Communication and Its Use in Mechanically Verifying a Biphase Mark Protocol

Participants: J Strother Moore

Contact name and address:

J Strother Moore
Computational Logic Inc.
1717 W. 6th St., Suite 290
Austin, TX 78703

(512) 322-9951
email moore@cli.com

Level of Effort: 3 man-months

Brief Description:

In this paper we present a formal model of asynchronous communication as a function in the Boyer-Moore logic. The function transforms the signal stream generated by one processor into the signal stream consumed by an independently clocked processor. This transformation ‘‘blurs’’ edges and ‘‘dilates’’ time due to differences in the phases and rates of the two clocks and the communications delay. The model can be used quantitatively to derive concrete performance bounds on asynchronous communications at ISO protocol level 1 (physical level). We develop part of the reusable formal theory that permits the convenient application of the model. We use the theory to show that a biphase mark protocol can be used to send messages of arbitrary length between two asynchronous processors. We study two versions of the protocol, a conventional one which uses cells of size 32 cycles and an unconventional one which uses cells of size 18. Our proof of the former protocol requires the ratio of the clock rates of the two processors to be within 3% of unity. The unconventional biphase mark protocol permits the ratio to vary by 5%. At nominal clock rates of 20MHz, the unconventional protocol allows transmissions at a burst rate of slightly over 1MHz. These claims are formally stated in terms of our model of asynchrony; the proofs of the claims have been mechanically checked with the Boyer-Moore theorem prover, NQTHM. We conjecture that the protocol can be proved to work under our model for smaller cell sizes and more divergent clock rates but the proofs would be harder. Known inadequacies of our model include that (a) distortion due to the presence of an edge is limited to the time span of the cycle during which the edge was written, (b) both clocks are assumed to be linear functions of time (i.e., the rate of a given clock is unwavering) and (c) reading ‘‘on an edge’’ produces a nondeterministically defined value rather than an indeterminate value. We discuss these problems.

Published articles and reports:

J Strother Moore, ‘‘A Formal Model of Asynchronous Communication and Its Use in Mechanically Verifying a Biphase Mark Protocol.’’ Technical Report 68, Computational Logic, Inc., August, 1991.

Name of application project: Checking correctness of mutex algorithm

Participants: Misao Nagayama and Carolyn Talcott

Contact name and address:

Carolyn Talcott
Computer Science Department
Stanford University
Stanford CA 94305
clt@sail.stanford.edu

Level of effort: One graduate student part time for approximately one year.

Brief description:

This was an exercise in checking an informal proof. It was carried out by a logic student who had no previous experience with computers, or provers, as a means of learning about using computers. We began with an informal proof of a Mutex Algorithm, given by Manna and Pnueli, based on their formalism for reasoning about concurrent programs. Minor modifications in data structures modelling program state were made to facilitate representation in the Nqthm logic. The Nqthm events followed closely the informal proof analysis. Some work was required to fill in details in some cases, and to find the right hints and lemmas.

Published articles and reports:

Nagayama, Misao and Talcott, Carolyn, ‘‘An NQTHM Mechanization of ‘An Exercise in the Verification of Multi-Process Programs’.’’ Technical Report STAN-CS-91-1370, Computer Science Department, Stanford University, 1991.

Name of application project: Verification of Cathedral hardware module library.

Participants: D. Verkest and J. Vandenbergh

Contact names and address: D. Verkest or L. Claesen, IMEC (VSDM), Kapeldreef 75, B-3001 Leuven, Belgium; email verkest%imec.imec.be@imec.be

Level of effort: 3 to 4 man years

Brief description:

The project is concerned with the formal verification with Nqthm of the functional correctness of parameterized hardware module libraries for use in high level silicon compilers. The libraries contain more than 25 modules of varying complexity ranging from simple functional building blocks which implement logical functions to a complex and highly optimized Booth multiplier module. The library contains both combinational and sequential blocks.

A library of bit vector functions has been developed together with a set of lemmas expressing useful properties of these functions. Abstraction functions and representation functions allow for switching between the bit vector domain and the domain of natural numbers and integers. The behavioral specification of the modules is modeled as a function in the logic using these bit vector functions. The behavior of the basic cells (leaf cells) is also modeled by functions in the logic (using the built-in logical connectives of the logic). The parameterized structure of the modules is described hierarchically in terms of the leaf cells. A proof of correctness is then constructed using Nqthm. Since all of the modules are parameterized in the word length of their input signals, the correctness proofs make use of Nqthm’s induction capabilities. During the development of the proofs the interactive proof checker (Pc-Nqthm) and Nqthm’s interactive break facilities (BREAK-LEMMA) are used to investigate the cause of proof failures.

Once the formal proof of a module is completed the possibility exists to automatically generate parameterized descriptions of its structure in a traditional hardware description language. These descriptions can be used by standard cell packages to generate (guaranteed correct) layout. In addition it is possible to perform net list comparison between instances of the module generated from the formal descriptions in the Boyer-Moore logic and the actual full-custom layout of the modules.

The library of bit vectors is also used to describe the semantics of operators in various hardware description languages in use at IMEC and it has been used to describe and verify the implementation of a division algorithm on an arithmetic logic unit with respect to the division on integers [2]. Some of the lemmas in this proof are realized with the interactive proof checker (Pc-Nqthm).

The main benefit of this project lies in a large increase in quality of the module library developed. The theorem prover approach forced us to meticulously check the consistency between implementation, specification and documentation of all aspects of the modules at all levels of hierarchy and abstraction. Many inconsistencies, including errors in functionality of the modules, were (and are still being) discovered using this approach. In addition the bit vector function library developed in the project can be used to check consistency between operators defined in the various hardware description languages at different levels of abstraction. Although the use of a theorem prover requires a large effort from a skilled user, we believe this effort is justified for the specific application domain of module libraries (or more in general any hardware/software library) because the effort can be written off over all the instances of the various modules which are generated.

Published articles and reports:

[1] D.Verkest, L.Claesen, H.De Man, ‘‘On the use of the Boyer-Moore theorem prover for correctness proofs of parameterized hardware modules,’’ in Formal VLSI Specification and Synthesis, Ed. L.Claesen, Elsevier Science Publishers (North Holland), 1990, pp.99-116

[2] D.Verkest, L.Claesen, H.De Man, ‘‘A proof of the non restoring division algorithm and its implementation on the Cathedral-II ALU,’’ Proc. of the workshop on Designing Correct Circuits, DCC-92, Lyngby, Denmark, 6-8 January 1992.

Name of application project: A Proven-Correct Compiler Front-End

Participants: Debora Weber-Wulff

Contact name and address: Debora Weber-Wulff, Institut fuer Informatik der FU Berlin, Nestorstr. 8-9, D-W-1000 Berlin 31

Level of effort: 2 1/2 years invested, probably 2 more years needed

Brief description:

The goal of this dissertation project is to prove an implementation for compiler front-ends correct with respect to a set of specifications using a mechanical verification system, the Boyer-Moore prover. The front-end consists of a scanner, a skeleton parser, a transformer from concrete to abstract syntax, and a parser table generator. The front-ends are for compilers that were developed for the ProCoS (ESPRIT BRA 3104: Provably Correct Systems) project. The fully verified front-ends will in the LISP subset employed by the rest of the compiler effort. The rest of the compiler, which compiles a representation for abstract syntax into Transputer code, has been proven correct by hand proofs.

One major benefit of using the prover is that one is made to very carefully examine what one is doing. Even the slightest presupposition that is not explicitly stated will make a proof fail or fail to terminate. The major drawback is the incredible learning curve associated with using the prover outside of Austin. It takes an enormous amout of time before one begins to feel comfortable proving even trivialities. Larger proof attempts without the support of persons experienced in using the prover are extremely difficult. It has been observed that one spends most of one’s time patiently coaxing this stubborn prover into assenting to the obvious.

In order for the prover to be more usable outside of Austin, there needs to be more of a body of little examples that illuminate perhaps one point about the use of the prover and more libraries available, so that one can build on "proven knowledge" and not have to begin every proof literally at GROUND-ZERO.

Published articles and reports:

‘‘Trip Report : Visit to Computational Logic, Inc., Austin, Texas’’ ProCoS Report. Kiel, February 1990.

‘‘Proof Movie : Proving the Add-Assign Compiler with the Boyer-Moore Prover’’. ProCoS Report. Kiel, July 1990. A thoroughly revised version is to appear in Formal Aspects of Computing.

‘‘The ‘Automated Proving and Term Rewriting’ Praktikum’’. Zusammen mit Karl-Heinz Buth. ProCoS Report. Kiel, February 1991.

‘‘Pass Collapsing : An Optimization Method for Compiler Proofs’’. ProCoS Report. Kiel, September 1991.

‘‘Proven Correct Front-end Specification’’. In: ESPRIT BRA 3104 ProCoS: Provably Correct Systems Final Report. Volume 3. Dines Bjorner (Ed.). In preparation.

----

1. J S. Moore, ‘‘A Formal Model of Asynchronous Communication and Its Use in Mechanically Verifying a Biphase Mark Protocol’’, Tech. report CLI Technical Report 68, Computational Logic, Inc., 1717 W. Sixth Street, Suite 290, Austin, TX 78703, August 1991.

2. R. S. Boyer and J S. Moore, ‘‘The Addition of Bounded Quantification and Partial Functions to A Computational Logic and Its Theorem Prover’’, Tech. report ICSCA-CMP-52, Institute for Computer Science, University of Texas at Austin, January 1988, To appear in the *Journal of Automated Reasoning*, 1988

3. R. S. Boyer, M. W. Green and J S. Moore, ‘‘The Use of a Formal Simulator to Verify a Simple Real Time Control Program’’, Technical Report ICSCA-CMP-29, University of Texas at Austin, 1982.

4. R. S. Boyer and J S. Moore, ‘‘A Verification Condition Generator for FORTRAN’’, in *The Correctness Problem in Computer Science,* R. S. Boyer and J S. Moore, eds., Academic Press, London, 1981.

5. R.S. Boyer, D. Goldschlag, M. Kaufmann, J S. Moore, ‘‘Functional Instantiation in First Order Logic’’, in *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy,* V. Lifschitz, ed., Academic Press, 1991, pp. 7-26.

6. D.M. Russinoff, ‘‘A Mechanical Proof of Quadratic Reciprocity’’, *Journal of Automated Reasoning,* Vol. 8, No. 1, 1992, pp. ???.

7. R. S. Boyer and J S. Moore, *A Computational Logic Handbook,* Academic Press, Boston, 1988.

8. R. S. Boyer and J S. Moore, *A Computational Logic,* Academic Press, New York, 1979.

9. R. S. Boyer and J S. Moore, ‘‘Proof Checking the RSA Public Key Encryption Algorithm’’, *American Mathematical Monthly,* Vol. 91, No. 3, 1984, pp. 181-189.

10. R. S. Boyer and J S. Moore, ‘‘A Mechanical Proof of the Turing Completeness of Pure Lisp’’, in *Automated Theorem Proving: After 25 Years,* W.W. Bledsoe and D.W. Loveland, eds., American Mathematical Society, Providence, R.I., 1984, pp. 133-167.

11. R. S. Boyer and J S. Moore, ‘‘A Mechanical Proof of the Unsolvability of the Halting Problem’’, *JACM,* Vol. 31, No. 3, 1984, pp. 441-458.

12. D.M. Russinoff, ‘‘A Mechanical Proof of Wilson’s Theorem’’, Masters Thesis, Department of Computer Sciences, University of Texas at Austin, 1983.

13. J S. Moore, ‘‘A Mechanical Proof of the Termination of Takeuchi’s Function’’, *Information Processing Letters,* Vol. 9, No. 4, 1979, pp. 176-181.

14. W. Bevier, *A Verified Operating System Kernel,* PhD dissertation, University of Texas at Austin, 1987.

15. A. Bronstein, *MLP: String-functional semantics and Boyer-Moore mechanization for the formal verification of synchronous circuits,* PhD dissertation, Stanford University, 1989.

16. A. Bronstein and C. Talcott, ‘‘String-Functional Semantics for Formal Verification of Synchronous Circuits, Report No. STAN-CS-88-1210’’, Tech. report, Computer Science Department, Stanford University, 1988.

17. A. Bronstein and C. Talcott, ‘‘Formal Verification of Synchronous Circuits based on String-Functional Semantics: The 7 Paillet Circuits in Boyer-Moore’’, *C-Cube 1989 Workshop on Automatic Verification Methods for Finite State Systems. LNCS 407,* 1989, pp. 317-333.

18. J S. Moore, ‘‘Piton: A Verified Assembly-Level Language’’, Tech. report CLI-22, Computational Logic, Inc., Austin, Tx, June 1988.

19. W.A. Hunt, Jr. and B. Brock , ‘‘A Formal HDL and its use in the FM9001 Verification’’, *Proceedings of the Royal Society,* 1992, to appear April 1992

20. R. S. Boyer and J S. Moore, ‘‘The Mechanical Verification of a FORTRAN Square Root Program’’, CSL Report, SRI International, 1981.

21. R. S. Boyer and J S. Moore, ‘‘MJRTY - A Fast Majority Vote Algorithm’’, Technical Report ICSCA-CMP-32, Institute for Computing Science and Computer Applications, University of Texas at Austin, 1982.

22. W.A. Hunt, Jr., *FM8501: A Verified Microprocessor*, PhD dissertation, University of Texas at Austin, 1985.

23. W.D. Young, ‘‘A Simple Expression Compiler: A Programming and Proof Example for an Nqthm Course’’, Tech. report Internal Note 210, Computational Logic, Inc., 1717 W. Sixth Street, Suite 290, Austin, TX 78703, November 1990.

24. M. Kaufmann, ‘‘Generalization in the Presence of Free Variables: A Mechanically-Checked Proof for One Algorithm’’, *Journal of Automated Reasoning*, Vol. 7, No. 1, March 1991.

25. M. Kaufmann, ‘‘An Extension of the Boyer-Moore Theorem Prover to Support First-Order Quantification’’, Tech. report CLI Technical Report 43, Computational Logic, Inc., 1717 W. Sixth Street, Suite 290, Austin, TX 78703, May 1989, To appear in J. of Automated Reasoning

26. M. Kaufmann, ‘‘A Simple Example for Nqthm: Modeling Locking’’, Tech. report Internal Note 216, Computational Logic, Inc., 1717 W. Sixth Street, Suite 290, Austin, TX 78703, February 1991.

27. M. Kaufmann, ‘‘An Example in Nqthm: Ramsey’s Theorem’’, Tech. report Internal Note 100, Computational Logic, Inc., 1717 W. Sixth Street, Suite 290, Austin, TX 78703, May 1991.

28. M. Kaufmann, ‘‘An Instructive Example for Beginning Users of the Boyer-Moore Theorem Prover’’, Tech. report Internal Note 185, Computational Logic, Inc., 1717 W. Sixth Street, Suite 290, Austin, TX 78703, April 1990.

29. M. Wilding, ‘‘Proving Matijasevich’s Lemma with a Default Arithmetic Strategy’’, *Journal of Automated Reasoning*, Vol. 7, No. 3, 1991, pp. 439-446.

30. M. Kaufmann, ‘‘An Integer Library for Nqthm’’, Tech. report Internal Note 182, Computational Logic, Inc., 1717 W. Sixth Street, Suite 290, Austin, TX 78703, March 1990.

31. W.R. Bevier, ‘‘A Library for Hardware Verification’’, Tech. report Internal Note 57, Computational Logic, Inc., 1717 W. Sixth Street, Suite 290, Austin, TX 78703, August 1988.

32. N. Shankar, ‘‘A Mechanical Proof of the Church-Rosser Theorem’’, Tech. report ICSCA-CMP-45, Institute for Computing Science, University of Texas at Austin, 1985.

33. N. Shankar, *Proof Checking Metamathematics*, PhD dissertation, University of Texas at Austin, 1986.

34. N. Shankar, ‘‘Towards Mechanical Metamathematics’’, *Journal of Automated Reasoning*, Vol. 1, No. 4, 1985, pp. 407-434.

35. M. Nagayama and C. Talcott, ‘‘An NQTHM Mechanization of ‘‘An Exercise in the Verification of Multi-Process Programs’’’’, Tech. report STAN-CS-91-1370, Computer Science Department, Stanford University, 1991.

36. R.S. Boyer and Y. Yu, ‘‘Automated Correctness Proofs of Machine Code Programs for a Commercial Microprocessor’’, Tech. report TR-91-33, Computer Sciences Department, University of Texas, Austin, November 1991.

37. Y. Yu, ‘‘Computer Proofs in Group Theory’’, *Journal of Automated Reasoning*, Vol. 6, No. 3, 1990.

38. R.S. Boyer and Y. Yu, ‘‘A Formal Specification of Some User Mode Instructions for the Motorola 68020’’, Tech. report TR-92-04, Computer Sciences Department, University of Texas, Austin, February 1992.

39. David Goldschlag, ‘‘A Mechanical Formalization of Several Fairness Notions’’, in *VDM ’91: Formal Software Development Methods*, S. Prehn and W.J. Toetenel, eds., Springer-Verlag Lecture Notes in Computer Science 551, 1991.

40. David M. Goldschlag, ‘‘Mechanically Verifying Safety and Liveness Properties of Delay Insensitive Circuits’’, *Computer Aided Verification 1991*, July 1991.

41. David M. Goldschlag, ‘‘Mechanizing Unity’’, in *Programming Concepts and Methods*, M. Broy and C. B. Jones, eds., North Holland, Amsterdam, 1990.

42. William D. Young, ‘‘A Mechanically Verified Code Generator’’, *Journal of Automated Reasoning*, Vol. 5, No. 4, December 1989, pp. 493-518, Also published as CLI Technical Report 30

43. Donald I. Good, Ann E. Siebert, William D. Young, ‘‘Middle Gypsy 2.05 Definition’’, Technical Report 59, Computational Logic, Inc., May 1990.

Index

Table of Contents