IN-61-CR

34007

OCIT

P-16

Computer Science Dept.
Stanford University
Stanford, CA 94305

Open Architectures for Formal Reasoning

and

Deductive Technologies For Software Development

Final Report
5/1/91-7/31/94
Grant no. NAG2-703

Principal Investigators

John McCarthy

Zohar Manna

Associate Investigators

Ian Mason

Amir Pnueli

Carolyn Talcott

Richard Waldinger

This project was organized as two separate subtasks: (i) *Open Architectures for Formal Reasoning*, and (ii) *Deductive Technologies for Software Development*. This final technical report consists of two sections of text summarizing the results for the two subtasks respectively.

(NASA-CR-197528) OPEN ARCHITECTURES FOR FORMAL REASONING AND DEDUCTIVE TECHNOLOGIES FOR SOFTWARE DEVELOPMENT Final Technical Report, 1 May 1991 - 31 Jul. 1994 (Stanford Univ.) 16 p

N95-16586

Unclas

G3/61 0034007

1 Open Architectures for Formal Reasoning

The objective of this project is to develop an open architecture for formal reasoning systems. One goal is to provide a framework with a clear semantic basis for specification and instantiation of generic components; construction of complex systems by interconnecting components; and for making incremental improvements and tailoring to specific applications. Another goal is to develop methods for specifying component interfaces and interactions to facilitate use of existing and newly built systems as "off the shelf" components, thus helping bridge the gap between producers and consumers of reasoning systems.

In this report we summarize results in several areas: our data base of reasoning systems; a theory of binding structures; a theory of components of open systems; a framework for specifying components of open reasoning system; and an analysis of the integration of rewriting and linear arithmetic modules in Boyer-Moore using the above framework.

1.1 Database

A database of existing automated reasoning systems is now available via anonymous ftp to sail.stanford.edu in the directory pub/clt/ARS. A README file there provides further details. Currently the entries consist of a brief description of the underlying logic, inference mechanisms, interaction capabilities, documentation, and major applications. They also contain information on availability and user support. The database will serve as a repository of information of interest to builders and users of automated reasoning systems. It provides a place to put pointers to both more detailed analyses, and eventually to rigorous specifications of interfaces and interactions. It will also provide a concrete basis for building consensus on common languages for describing shared data structures and semantics of systems.

The database will soon be put into html format to take advantage of the WWW browsing tools. There are currently links to the database from numerous other Web pages including pages related to: formal methods; automated reasoning; and programming language theory.

We are currently looking for ways to make the information more useful. One possible step in this direction is suggested by email conversations with Landauer at AeroSpace. Namely designing wrappings for database entries. Wrappings provide computer readable information about components that facilitate their off-the-shelf use. AeroSpace is developing tools for using wrapping in design, simulation, and configuration of complex systems.

1.2 Binding Structures

A theory of binding structures has been developed to facilitate representation and manipulation of symbolic structures such as programs, logical formulae, derivations, specifications and modules. This work in reported in [14, 13, 17]. Binding structures enrich traditional abstract syntax trees by providing support for representing binding mechanisms and structures with schematic variables. The goal of this work is to establish a common core for building tools such as theorem provers, transformers, static analyzers, evaluators, rewriters, etc. that manipulate symbolic structures. Binding structures solve problems of variable name conflict and renaming, and provide a means for manipulating occurrences of structures. They incorporate the notion of syntactic context. This allows for expression of schemata within the language rather than as meta-expressions. Instantiating schematic variables is a mechanism for capturing free variables, in contrast to substitution for free variables, which avoids capture. Our focus has been on finding clear descriptions of operations on symbolic structures. This has led to such notions as parameterized homomorphism and uniform reduction rules. A parameterized homomorphism is a simple structure walking tool that preserves binding relations even in the presence of holes. Uniform reduction rules include traditional term rewriting, combinatory rewrite systems, reduction rules for various lambda calculi, and program transformation rules. They can be applied to structures with holes (schemata), and rule application commutes with hole filling (schema instantiation). This is the key property for being able to reason using symbolic computation. The notion of unification of algebraic terms extends naturally to binding structures, using substitution for free and externally bound variables. Unification with respect to filling holes corresponds to unification of higher-order patterns, a decidable fragment of higher-order unification. In both cases there is a most general unifier. This work has been used in combination with work of Feferman [6] as the basis for implementation of a meta-logic for specifying and reasoning about logics and programming languages [10]. The results will be used in developing tools for specifying and integrating reasoning systems, within the framework for reasoning systems described below.

1.3 A Theory of Open System Components

Our approach is based on the *Actor model* [9, 1, 4]. Actors provide an abstract model of an open distributed computer system. An actor is an encapsulated object that communicates with other actors through asynchronous point-to-point message passing. New actors may be created dynmaically, and actor addresses may be communicated in messages. An important aspect of the actor model is the fairness requirement: message delivery is guaranteed, and individual actor computations are guaranteed to progress.

It is important to note that our methodology is not dependent on any specific programming notation. Instead, we assume only that the three basic operations: message sending, actor creation, and change of local state. are in some way incorporated. In particular, the actor operators may be used to "wrap" existing sequential programs, thus serving as an interconnection language.

To study the semantics of components of open systems we have defined an actor language that is an extension of a simple functional language [3, 2]. Actors can be viewed as individual actor systems, and systems can be composed to form larger systems. Our actor systems are *open*—they can send messages outside the system, and designated actors can receive messages from outside the system. Focusing on systems as well as individual actors has several advantages. It solves the problem of compositionality - systems of actors may be composed. It provides a clean interface between the component and its environment. Since our actor language provides a natural means of defining a wide variety of operations for combining components, the theory of actors and actor systems provides a means of expressing properties of such operations. As such it can be thought of as an algebra of components.

The semantics for actor systems is given by a transition relation on open configurations. A configuration contains a collection of actors, messages together with an interface specifying external actor names, and receptionist names. The semantics of our language extends that of the embedded functional language in such a way that the equational theory of the functional language is preserved. The fairness assumption fo the actor model makes some aspects of reasoning more complicated, but simplifies others. Many intuitively correct equations fail in the absence of fairness. We have shown that in presence of fairness the three standard notions of observational equivalence for non-deterministic/concurrent computation collapse to two. A variety of laws for observational equivalence have been established and several general proof techniques have been developed for establishing such laws. In addition to developing the basic theory of for our actor language, there is work in progress to formalize the semantics in the PVS proof development system [12] and develop mechanically checked proofs of the main theorems. This is a first step towards building of a set of tools for computer aided specification and design of open distributed systems. The formalization work has already resulted in several improvements in the structure of the semantic model. Work is in progress developing a calculus of component of open systems based on our actor model. This includes abstracting from the particular choice of language and developing an interaction semantics based on patterns of message passing.

1.4 Open Mechanized Reasoning Systems.

The ultimate goal of this work is to provide a framework and a methodology which will allow users, and not only system developers, to construct complex reasoning systems by composing existing modules, or to add new modules to existing systems, in a "plug and play" manner. These modules and systems might be based on different logics; have different domain models; use different vocabularies and data structures; use different reasoning strategies; and have different interaction capabilities. The paper [7] reports two recent contributions towards this goal. First, it proposes a general architecture for a class of reasoning modules and systems called *Open Mechanized Reasoning Systems* (*OMRSs*). An OMRS has three components: a *reasoning theory* component which is the counterpart of the logical notion of formal system, a *control* component which consists of a set of inference strategies, and an *interaction* component which provides an OMRS with the capability of interacting with other systems, including OMRSs and human users. Second, it develops the theory underlying the reasoning theory component. This development is motivated by an analysis of state of the art systems. The resulting theory is then validated by using it to describe the integration of the linear arithmetic module into the simplification process of the Boyer-Moore system, NQTHM.

A reasoning theory is given by a sequent system and a set of rules. Associated to each reasoning theory is a set of reasoning structures. These structures represent fragements of deductions. They may be schematic (holes left to be filled in) and/or provisional (with unsolved constraints). Only certain general features of sequents and rules are needed to describe the notions of reasoning structure and derivation associated to a reasoning theory, and the operations for constructing reasoning structures. These are abstracted in the notions of *sequent system* and abstract rule. This allows us to decouple the definitions of reasoning structure and derivation from the details of any specific notation for presenting sequent systems and rules.

An importanat aspect that is currently missing from the OMRS framework is a notion of model or semantics. In [15], we examined the notion of logical system [11] as a possible framework for specifying the semantic behavior of components of automated reasoning systems and for describing sound interconnections between these components. Of particular importance are maps between logical systems. They provide translations needed for communication between OMRS. They also provide a basis for use of systems as components in heterogeneous combinations and for developing a calculus of reasoning theory modules.

Another important direction of future work is to include mechanisms for encapsulating and sharing structures, and for interaction. Preliminary ideas and results on developing the interaction aspect of OMRS are reported in [16]. Interaction semantics will be based both on logical systems work and on our work on components of open distributed systems (see above). A full logical system [11] also contains a notion of model and semantics for sequents. Extending the notion of OMRS to include semantics is an important topic for future work. Another important direction of future work is to include mechanisms for encapsulating and sharing structures, and for interaction. Preliminary ideas and results on developing the interaction aspect of OMRS are reported in [16]. Interaction semantics will be based both on logical systems work and on our work on components of open distributed systems (see above).

1.5 Analysis of Integration in the Boyer-Moore Prover

As a test of the reasoning theory framework, we have carried out an analysis of the Boyer-Moore prover, NQTHM. We focused on the integration of the linear-arithmetic module as described in [5]. The analysis is sketched in Part III of [7]. A more detailed description will be provided in a forthcoming report [8].

We chose this example as our first benchmark for several reasons. NQTHM is a mature system that is highly tuned and has a large user community. A better understanding of how it works is by itself of considerable interest), NQTHM is thoroughly documented, and it constitutes one of the most challenging case studies we could think of.

One of the main difficulties in the integration of a new module into a tightly coded system like NQTHM is that the existing procedures must be modified to generate, manipulate and propagate the information needed or generated by the new module. For example, in the case of integration of linear arithmetic, the local context information is represented in two ways: as typeset information and as polynomial information. In addition, the linear arithmetic module generates additional assumptions and dependency information that the rewriter must propagate. One of our main goals here is to show how the extra information and modifications can be isolated inside the definition of the sequent system and rules of the modified system. The methodology we use is the following:

(1) Specification of the original system;

(2) Specification of the module to be added;

(3) Refinement of the specification of the original system to incorporate the additional information passed to and from the new module.

(4) "Gluing together" of the new module and the modified system, which might require the addition of new bridge rules.

We conclude our analysis of NQTHM by giving some example reasoning structures representing NQTHM deductions. We do this both to give some realistic examples, and to suggest how this methodology can be applied to provide NQTHM with the (presently missing) capability of producing proof structures.

References

[1] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, Cambridge, Mass., 1986.

[2] G. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. A foundation for actor computation, 199? to appear.

[3] G. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. Towards a theory of actor computation. In *The Third International Conference on Concurrency Theory (CONCUR '92)*, volume 630 of *Lecture Notes in Computer Science*, pages 565–579. Springer Verlag, August 1992.

[4] Gul Agha. Concurrent object-oriented programming. *Communications of the ACM*, 33(9):125–141, September 1990.

[5] Robert S. Boyer and J. Strother Moore. Integrating decision procedures into heuristic theorem provers: A case study of linear arithmetic. Technical Report ICSCA-CMP-44 , Institute for Computing Science, University of Texas at Austin, 1985.

[6] S. Feferman. Finitary inductively presented logics. In *Logic colloquium 88*, pages 191–220. North-Holland, 1988.

[7] F.. Giunchiglia and C. L. Pecchiari, P. Talcott. Reasoning theories: Towards an architecture for open mechanized reasoning systems, November 1994.

[8] F. Giunchiglia, P. Pecchiari, and C. Talcott. An analysis of the reasoning structures and rules underlying the integration of linear arithmetic in to the Boyer-Moore prover, in preparation.

[9] C. Hewitt. Viewing control structures as patterns of passing messages. *Journal of Artificial Intelligence*, 8(3):323–364, 1977.

[10] S. Mathews. Metatheory and reflection. In *Proceedings of CADE-12 workshop on Correctness and Metatheoretic Extensibility of Automated Reasoning Systems*, pages 4–5, 1994.

[11] José Meseguer. General logics. In H.-D. Ebbinghaus et al., editor, *Logic Colloquium'87*, pages 275–329. North-Holland, 1989.

[12] S. Owre, J. Rushby, and N. Shankar. PVS: A prototype verification system, 1992.

[13] C. L. Talcott. Binding structures. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation.* Academic Press, 1991.

[14] C. L. Talcott. Towards a theory of binding structures. In *Second International Conference on Algebraic Methodology and Software Technology, AMAST*, 1991. to appear in LNCS, full version to appear in TCS.

[15] C. L. Talcott. Towards a framework for specifying components of automated reasoning systems: A report on work in progress. In *TTCP XTP-1 Workshop on Effective Use of Automated Reasoning Technology in System Development (EUARTSD)*, 1992.

[16] C. L. Talcott. Reasoning specialists should be logical services, not black boxes. In *Proceedings of CADE-12 workshop on Theory Reasoning in Automated Deduction*, pages 1-6, 1994.

[17] C.L. Talcott. A theory of binding structures and its applications to rewriting. *Theoretical Computer Science*, 112:99–143, 1993.

2 Deductive Technology for Software Development

Our research concentrated on the following topics:

2.1 Automated Deduction

New Deduction Rules [19]

We developed inferences rules to achieve abbreviated proofs in theories possessing certain monotonicity properties, such as transitivity and substitutivity. These rules are extensions of the relation replacement and relation matching rules.

The relation replacement rule was extended to allow the formula undergoing replacement to experience additional repercussions. The extended rule subsumes the negative paramodulation rule and gives a cleaner treatment to relation strengthening, which was handled in an *ad hoc* way by the original relation-replacement rule.

The relation replacement rule, when used alone, is incomplete. The relation matching rule was extended to match distinct predicate and function symbols and to allow the reordering of their arguments. The new rules allow shorter proofs and a smaller search space than if monotonicity properties are represented explicitly.

Temporal Deduction [21]

We developed a deductive system for predicate temporal logic with induction. This deductive system is relatively complete.

Representing temporal operators by first-order expressions enables temporal deduction to use the already developed techniques of first-order deduction. But when translating from temporal logic to first-order logic is done indiscriminately, the ensuing quantifications and comparisons of time expressions encumber formulas, hindering deduction. In our deductive system, translation occurs more carefully, via *reification* rules. These rules paraphrase selected temporal formulas as nontemporal first-order formulas with *time annotations*. This process of time reification suppresses quantifications (the process is analogous to quantifier skolemization) and uses addition instead of complicated combinations of comparisons. Some ordering conditions on arithmetic expressions can arise, but these are handled automatically by a special-purpose unification algorithm plus a decision procedure for Presburger arithmetic.

2.2 Program Synthesis

Realizability and Synthesis [1]

We present two algorithms: a realizability-checking algorithm and a synthesis algorithm. Given a specification of reactive asynchronous modules expressed in propositional ETL (Extended Temporal Logic), the realizability-checking algorithm decides whether the specification has an actual implementation, under the assumptions of a random environment and fair execution. It also creates a structure which can then be transformed by the synthesis algorithm into a program, represented as a labeled finite automaton. Unlike previous approaches, the realizability-checking algorithm can handle fairness assumptions. The realizability-checking algorithm is incremental and it directly manipulates formulas in linear temporal logic without having to transform into a branching-time logic or other representations.

The Deductive Synthesis of Computer Programs [18]

Program synthesis is the systematic derivation of a computer program to meet a given specification. Here the specification is a general description of the purpose of the desired program, while the program is a detailed description of a method for achieving that purpose. The particular emphasis of this project is on the development of deductive techniques, i.e., techniques based on theorem proving, for program synthesis. These techniques are amenable to automatic implementation, but may be used interactively or for the formal explication of informal derivations discovered by hand.

Some achievements of the project are as follows:

- Synthesis of a class of recursive unification algorithms (algorithms for finding a common instance of two expressions).
- Development of a situational logic for the synthesis of nonapplicative programs (programs with side effects).
- Introduction of deduction rules giving accelerated performance for relations of special importance to program synthesis (such as equality and ordering relations).
- Synthesis of a class of real-number algorithms employing the binary-search technique (such as binary-search square-root algorithm).
- Completion of a survey summarizing on a more elementary level the deductive approach to program synthesis.

2.3 Temporal Logic

A Hierarchy of Temporal Properties [3]

We proposed a classification of temporal properties into a hierarchy which refines the known *safety-liveness* classification of properties. The classification is based on the different ways a property of finite computations can be extended into a property of infinite computations.

This hierarchy was studied from three different perspectives, which were shown to coincide. We examined the cases in which the finitary properties, and the infinitary properties extending them, are unrestricted, specifiable by temporal logic, or specifiable by predicate automata. The unrestricted view leads also to a topological characterization of the hierarchy as corresponding to the lowest two levels in the Borel hierarchy.

For properties that are expressible by temporal logic and predicate automata, we provided a syntactic characterization of the formulas and automata that specify properties of the different classes.

Corresponding to each class of properties, we presented a proof principle that is adequate for proving the validity of properties in that class.

2.4 Reactive Systems

Temporal Proof Methodology for Reactive Systems [15]

We developed a minimal proof theory which is adequate for proving the main important temporal properties of reactive programs. The properties we consider consist of the classes of *invariance*, *response*, and *precedence* properties. For each of these classes we present a small set of rules that is complete for verifying properties belonging to this class. We illustrate the application of these rules on several examples. We discuss concise presentations of complex proofs using the devices of *transition tables* and *proof diagrams.*

Verification Diagrams [17]

Most formal approaches to the verification of temporal properties of reactive programs infer temporal conclusions from verification conditions that are state formulas, i.e., contain no temporal operators. These proofs can often be effectively presented by the use of *verification diagrams.* We introduced verification diagrams as a tool for proving various temporal properties.

Beginning with safety properties, we present *wait-for* and *invariance* diagrams for proving wait-for (precedence) and invariance formulas. Proceeding to liveness properties, we present verification diagrams for response properties that require a bounded number of helpful steps (*chain* diagrams) and response properties that require an unbounded number of helpful steps (*rank* diagrams).

Additional types of diagrams are proposed for handling response properties for parameterized programs and response properties that rely on the full spectrum of fairness requirements, including compassionate helpful transitions.

2.5 Real-time Systems

Temporal Proof Methodologies for Real-time Systems [5],[6]

We extended the specification language of temporal logic, the verification framework, and the underlying computational model to deal with real-time properties of concurrent and reactive systems. A global, discrete, and asynchronous clock is incorporated into the model by defining the abstract notion of a real-time transition system as a conservative extension of traditional transition systems: qualitative fairness requirements are replaced (and superseded) by quantitative lower-bound and upper-bound real-time requirements for transitions.

We showed how to model programs that communicate either through shared variables or by message passing, and how to represent the important real-time constructs of priorities (interrupts), scheduling, and timeouts in this framework.

Two styles for the specification of real-time properties were presented. The first style uses bounded versions of the temporal operators; the real-time requirements expressed in this style are classified into *bounded-invariance* and *bounded-response* properties. The second specification style allows explicit references to the current time through a special clock variable.

Corresponding to the two styles of specification, we developed two very different proof methodologies for the verification of real-time properties expressed in these styles. For the *bounded-operators* style, we provided proof rules to establish lower and upper real-time bounds for response properties of real-time transition systems. For the *explicit-clock* style, we exploited the observation that, when given access to the clock, every real-time property can be reformulated as a safety property, and use the standard temporal rules for establishing safety properties.

Verification of Real-Time Systems [7]

We extend the specification language of temporal logic, the corresponding verification framework, and the underlying computational model to deal with real-time properties of reactive systems. The abstract notion of timed transition systems generalizes traditional transition systems conservatively: qualitative fairness requirements are replaced (and superseded) by quantitative lower-bound and upper-bound timing constraints on transitions. This framework can model real-time systems that communicate either through shared variables or by message passing and handle real-time issues such as timeouts, process priorities (interrupts), and process scheduling.

We exhibit two styles for the specification of real-time systems. While the first approach uses time-bounded versions of the temporal operators, the second approach allows explicit references to time through a special clock variable. Corresponding to the two styles of specification, we present and compare two different proof methodologies for the verification of timing requirements that are expressed in these styles. For the *bounded-operator* style, we provide a set of proof rules for establishing bounded-invariance and bounded-response properties of timed transition systems. This approach generalizes the standard temporal proof rules for verifying invariance and response properties conservatively. For the *explicit-clock* style, we exploit the observation that every time-bounded property is a safety property and use the standard temporal proof rules for establishing safety properties.

Compositional Verification of Real-time Systems [4]

We developed a compositional proof system for the verification of real-time systems. Real-time systems are modeled as timed transition modules, which explicitly model interaction with the environment and may be combined using composition operators.

Composition rules are devised such that the correctness of a system may be determined from the correctness of its components. These proof rules are demonstrated on Fischer's mutual exclusion algorithm, for which mutual exclusion and bounded response are proven.

2.6 Hybrid Systems

Specification and Verification of Hybrid Systems [9],[13]

We developed a framework for the formal specification and verification of *timed* and *hybrid* systems. For timed systems we proposed a specification language that refers to time only through *age* functions which measure the length of time that a given formula has continuously been true.

We then considered hybrid systems, which are systems consisting of a non-trivial mixture of discrete and continuous components, such as a digital controller that controls a continuous environment, control of process and manufacturing plants, guidance of transport systems, and robot planning. The proposed framework extends the temporal logic approach which has proven useful for the formal analysis of discrete systems such as reactive programs. The new framework consists of a semantic model for hybrid time, the notion of *phase transition systems*, which extends the formalism of discrete transition systems, an extended version of Statecharts for the specification of hybrid behaviors, and an extended version of temporal logic that enables reasoning about continuous change.

Models for Reactivity [14]

A hierarchy of models that captures realistic aspects of reactive, real-time, and hybrid systems was introduced. On the most abstract level, the qualitative (non-quantitative) model of *reactive systems* captures the temporal precedence aspect of time. A more refined model is that of *real-time systems*, which represents the metric aspect of time. The third and most detailed model is that of *hybrid systems*, which allows the incorporation of *continuous* components into a reactive system.

For each of the three levels, we developed a computational model, a requirement specification language based on extensions of temporal logic, system description languages based on Statecharts and a textual programming language, proof rules for proving validity of properties, and examples of such proofs.

Hybrid Temporal Logic [8]

We propose a methodology for the specification, verification, and design of hybrid systems. The methodology consists of the computational model of *Concrete Phase Transition Systems* (CPTS), the specification language of *Hybrid Temporal Logic* (HTL), the graphical system description language of *Hybrid Automata*, and a proof system for verifying that hybrid automata satisfy their HTL specifications.

The novelty of the approach lies in the continuous-time logic, which allows specification of both point-based and interval-based properties and provides direct references to derivatives of variables, and in the proof system that supports verification of point-based and interval-based properties. The proof rules demonstrate that sound and convenient induction rules can also be established for continuous-time logics and are not necessarily restricted to discrete logics. The proof rules are illustrated on several examples.

Design of Controlled Systems [22]

We propose a conceptual framework to support specification, design and verification of programs controlling physical systems. We introduce a computational model that represents the controller capabilities and distinguishes between synchronous and phase transitions. A graphical system description language is proposed that we believe is readily accessible to control engineers. We formalize the notion of control strategy in controller design.

2.7 Computer-Aided Verification Systems

TABLEAU:- An Interactive Graphic Deductive System

We have collaborated with a software-development team in developing an interactive theorem prover, TABLEAU, based on the deductive-tableau framework. Implemented on an Apple Macintosh computer, the system uses a graphical interface to communicate with the user. Rather than relying on the keyboard, the user may select with a mouse which step in the proof to attempt next. Although directing the proof is the responsibility of the user, the system intervenes if the user attempts an illegal step. The system can construct proofs in propositional and predicate logic, in theories of the nonnegative integers, trees, and tuples, and in new theories introduced by the user. The system complements the textbook [20] and is available for classroom use.

The system has a combination of features lacking elsewhere. In particular,

- It can handle theorems with both universal and existential quantifiers.
- It can produce proofs by mathematical induction, including well-founded induction.
- It has special provisions for reasoning about equality. Furthermore, the convenient interface of the system makes it far easier to use in constructing a detailed proof than it is to prove the same theorem by hand, a feature unfortunately not shared with many systems.

The system has been augmented in several directions:

- Introduced the capability of adding new deduction rules. This would facilitate the application of the system to new theories.
- Introduced a facility for extracting information from proofs. This information could be a program, a plan, or a database transaction.
- Allowed the gradual automation of the system. In particular, we would like to automate certain routine and repetitive aspects of the proof process.

STeP: the Stanford Temporal Prover [10], [11]

STeP (the Stanford Temporal Prover) is a system to support computer-aided verification of reactive systems based on their temporal specifications. Unlike most systems for temporal verification, STeP does not concentrate solely on finite-state systems. It combines model checking with algorithmic deductive methods (decision procedures) and interactive deductive methods (theorem proving). The user is expected to interact with the system and provide, whenever necessary, top-level guidance in the form of auxiliary invariants for safety properties, and well-founded measures and intermediate assertions for progress properties. In short, STeP has been designed with the objective of combining the expressiveness of deductive methods with the simplicity of model checking.

Development efforts have been focused, in particular, in the following areas.

First, in addition to the textual language of temporal logic, the system supports a structured visual language of *verification diagrams* ([16]) for guiding proofs. Verification diagrams allow the user to construct proofs hierarchically, starting from a high-level, intuitive proof sketch and proceeding incrementally, as necessary, through layers of greater detail.

Second, the system implements powerful techniques (algorithmic and heuristic) for automatic *invariant generation.* Deductive verification in the temporal framework almost always relies heavily on finding, for a given program and specification, suitably strong invariants and intermediate assertions. The user can typically provide an intuitive, high-level invariant, from which the system derives stronger, *top-down invariants.* Simultaneously, *bottom-up invariants* are generated automatically by analyzing the program text. By combining these two methods, the system can often deduce sufficiently detailed invariants to carry through the entire verification process.

Finally, the system provides a built-in facility for automatically checking a large class of first-order and temporal formulas, based on simplification methods, term rewriting, and decision procedures. This degree of automated deduction is sufficient to handle most of the verification conditions that arise during the course of deductive verification — and the few conditions that are not solved automatically correspond to the critical steps of manually constructed proofs, where the user is most capable of providing guidance.

Although the system is in an early stage of development, many of the examples in the Manna-Pnueli textbook ([17]) have already been automatically verified using STeP.

2.8 Ph.D. Thesis

Compositional Verification [2]

This thesis presents a compositional methodology for the verification of reactive and real-time systems. The correctness of a given system is established from the correctness of the system's components, each of which may be treated as a system itself and further reduced. When no further reduction is possible or desirable, global techniques for verification may be used to verify the bottom-level components.

Transition modules are introduced as a suitable compositional model of computation. Various composition operations are defined on transition modules, including parallel composition, sequential composition, and iteration. A restricted assumption-guarantee style of specification is advocated, wherein the environment assumption is stated as a restriction on the environment's next-state relation. Compositional proof rules are provided in accordance with the safety-progress hierarchy of temporal properties.

The compositional framework is then extended naturally to real-time transition modules and discrete-time metric temporal logic.

2.9 Books

Reactive Programs: Temporal Specification [12]

Reactive programs are programs whose role is to maintain an on-going interaction with their environment, rather than to produce some computational result on termination. Programs belonging to this class are usually described by some of the attributes: concurrent, distributed, real-time, and typical examples of such programs are: embedded programs, communication networks, control programs of industrial plants, operating systems, real-time programs, etc. Clearly, the correct and reliable construction of such programs is one of the most challenging programming tasks existing today.

Due to the special character of these programs, the formal approach to their specification and development must be based on the study of their *behavior* rather than on the function or relation they compute, an approach which has been adequate for computational and sequential programs.

We developed formal methods for the specification, verification and development of reactive programs, based on the formalism of *temporal logic* that has been specifically developed to reason about behaviors of reactive programs.

Among the topics we investigated are:

- Modeling reactive programs as fair transition systems.
- The language of temporal logic and its usage for the specification of program properties and system requirements.
- A classification of specifications into the classes of *safety* properties, *responsiveness* properties, and so on.

Deductive Foundations [20]

The research papers in which we have presented the deductive approach to program synthesis have been addressed to the usual academic readers of the scholarly journals. In an effort to make this work accessible to a wider audience, including computer science undergraduates and programmers, we have developed a more elementary treatment in the form of a book, *The Deductive Foundations of Computer Programming*, Addison-Wesley.

The text includes some novel research results, including

- theories of integers, tuples, and trees which are particularly well suited to theorem-proving and program-synthesis applications;
- a nonclausal version of skolemization;
- a treatment of mathematical induction in the deductive-tableau framework.

Reactive Systems: Temporal Verification [17]

We studied in detail the proof methodologies for verifying temporal properties of reactive systems. Appropriate proof principles were presented for each class of temporal *safety* and *progress* properties.

We developed proof principles for the establishment of *safety* properties. We showed that essentially there is only one such principle for safety proofs, the invariance principle, which is a generalization of the method of intermediate assertions. We also indicated special cases under which these assertions can be found algorithmically.

We illustrate the application of these rules on many examples. We suggest concise presentations of complex proofs using the devices of *transition tables* and *verification diagrams.*

Among the topics we investigated are:

- Methodologies for formal verification of the safety properties of a reactive program, and development approaches derived from them.
- Methodologies for formal verification of the responsiveness properties of a reactive program, and development approaches derived from them.
- Finite-state programs and associated automatic verification tools for them.

References

[1] A. Anuchitanukul and Z. Manna. Realizability and Synthesis of Reactive Modules. In 6th Conference on *Computer Aided Verification*, Lecture Notes in Computer Science 818, Springer-Verlag, 1994, pp. 156–168.

[2] E. Chang. Compositional Verification of Reactive and Real-Time systems. Ph.D. Thesis, Computer Science Department, Stanford University, 1994.

[3] E. Chang, Z. Manna, and A. Pnueli. Characterization of temporal property classes. In Proc. 19th Int. Colloq. Aut. Lang. Prog., Lect. Notes in Comp. Sci. 623, Springer-Verlag, pp. 474–486, 1992.

[4] E. Chang, Z. Manna and A. Pnueli. Compositional Verification of Real-Time Systems. Symposium on *Logic in Computer Science*, Paris, Lecture Notes in Computer Science, Springer-Verlag, pp. 458–465. 1994.

[5] T. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Proceedings of the REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lect. Notes in Comp. Sci.*, pages 226–251. Springer-Verlag, 1992.

[6] T. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In W. Kuich, editor, *Proc. 19th Int. Colloq. Aut. Lang. Prog.*, volume 623 of *Lect. Notes in Comp. Sci.*, pages 545–558. Springer-Verlag, 1992.

[7] T. Henzinger, Z. Manna, and A. Pnueli. Temporal Proof Methodologies for Timed Transition Systems. Information and Computation journal, Vol. 112, No. 2, pp. 273–337, 1994.

[8] A. Kapur, T. Henzinger, Z. Manna, and A. Pnueli. Proving Properties of Hybrid Systems. International Symposium on *Formal Techniques in Real Time and Fault Tolerant Systems*, Lecture Notes in Computer Science, Springer-Verlag, 1994.

[9] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Proceedings of the REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lect. Notes in Comp. Sci.*, pages 447–484. Springer-Verlag, 1992.

[10] Z. Manna. Beyond Model Checking. 6th Conference on *Computer Aided Verification*, Lecture Notes in Computer Science, Springer-Verlag, 1994.

[11] Z. Manna, N. Bjorner, A. Browne, E. Chang, M. Colon, L. de Alfaro, H. Devarajan, H. Sipma, and T. Uribe. *STeP: Stanford Temporal Prover*, Computer Science Department, Stanford University, 1994.

[12] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, New York, 1991.

[13] Z. Manna and A. Pnueli. Time for concurrency. In *INRIA's 25th Anniversary Volume*, volume 653 of *Lect. Notes in Comp. Sci.*, pages 129–153. Springer-Verlag, 1992.

[14] Z. Manna and A. Pnueli. Models for reactivity. *Acta Informatica*, Vol. 30, pp. 609–678. 1993.

[15] Z. Manna and A. Pnueli. A temporal proof methodology for reactive systems. In *Program Design Calculi*, NATO ASI Series, Series F: Computer and System Sciences. Springer-Verlag, Vol. 118, 1993.

[16] Z. Manna and A. Pnueli. Temporal Verification Diagrams. International Symposium on *Theoretical Aspects of Computer Software*, Lecture Notes in Computer Science 789, Springer-Verlag, pp. 726–765, 1994.

[17] Z. Manna and A. Pnueli, Temporal Verification of Reactive Systems. Springer-Verlag, New York (to appear, 1995).

[18] Z. Manna and R. Waldinger. Fundamentals of deductive program synthesis. *IEEE Trans. Software Engin.*, 18(8):674–704, 1991.

[19] Z. Manna and R. Waldinger. The special-relation rules are incomplete. In *Proc. of the 11th Conf. on Automated Ded.*, volume 607 of *Lect. Notes in Comp. Sci.*, pages 492–506. Springer-Verlag, 1992.

[20] Z. Manna and R. Waldinger. *The Deductive Foundations of Computer Programming.* Addison-Wesley, 1993.

[21] H. McGuire, Z. Manna and R. Waldinger. Annotation-Based Deduction in Temporal logic. First International Conference on Temporal Logic, Lecture Notes in Computer Science, Springer-Verlag, 1994.

[22] H.B. Sipma and Z. Manna, Specification and Verification of Controlled Systems. International Symposium on *Formal Techniques in Real Time and Fault Tolerant Systems*, Lecture Notes in Computer Science, Springer-Verlag, 1994.

```
DOC #: 34007        ABSTRACT SCREEN       PAGE: 1    ** WORDS NOT FOUND *

Open architectures for formal reasoning and deduct
ive technologies for software development


ABQ: NC                                                                   CIN: SAF
ABA: Author                                                               KIN: JXP
                                                                          AIN:
The objective of this project is to develop an     |
open architecture for formal reasoning systems.    |
One goal is to provide a framework with a clear    |
semantic basis for specification and instantiation | INSTANTIATION
 of generic components; construction of complex    |
systems by interconnecting components; and for     |
making incremental improvements and tailoring to   |
specific applications. Another goal is to develop  |
methods for specifying component interfaces and    |
interactions to facilitate use of existing and     |
newly built systems as 'off the shelf' components, |
 thus helping bridge the gap between producers and |
 consumers of reasoning systems. In this report we |
 summarize results in several areas: our data base |
 of reasoning systems; a theory of binding         |
structures; a theory of components of open         |
systems; a framework for specifying components of  |
open reasoning system; and an analysis of the      |
integration of rewriting and linear arithmetic     | REWRITING
modules in Boyer-Moore using the above framework.  |
                                                   |
                                                   |
                                                   |
                                                   |
                                                   +
                                                   +
___________________________________________________+
___________________________________________________
___________________________________________________

  PF1=ABA LIST; PF2=RESET; PF3=SIGNON; PF4=RELEASE FROM SUBQ; PF5=SELECTION;
  PF6=SUBQUEUE; PF7=STORE ABSTRACT; PF8=MAI; PF10=SEND TO 'MAIQ';
  PF14=PREVIOUS PAGE; PF15=NEXT PAGE; PF19=TITLE-EXT; PF20=INDEX TERMS
4B•                A                                        =-•PC LINE 11 COL 2
```

```
DOC NUMBER: 34007          INDEXING: SUBJECT/TERMS SCREEN          CIN: SAF
    TITLE: Open architectures for formal reasoning and deduct      KIN: JXP
           ive technologies for software development               AIN:
         MAJOR TERMS:                                   SWITCH
      1: COMPUTER PROGRAMMING______________________ _
      2: SOFTWARE ENGINEERING______________________ _
      3: COMPLEX SYSTEMS___________________________ _
      4: DISTRIBUTED PROCESSING____________________ _
      5: ARCHITECTURE (COMPUTERS)__________________ _
      6: SEMANTICS_________________________________ _
      7: INFORMATION TRANSFER______________________ _
      8: SOFTWARE TOOLS____________________________ _
      9: HUMAN-COMPUTER INTERFACE__________________ _
     10: COGNITION_________________________________ _
     11: MATHEMATICAL LOGIC________________________ _
     12: __________________________________________ _
     13: __________________________________________ _
     14: __________________________________________ _
     15: __________________________________________ _
         MINOR TERMS:
      1: DATA BASES________________________________ _
      2: MODULES___________________________________ _
      3: AUTOMATA THEORY___________________________ _
      4: INTERNETS_________________________________ _
      5: __________________________________________ _
      6: __________________________________________ _
      7: __________________________________________ _
      8: __________________________________________ _
      9: __________________________________________ _
     10: __________________________________________ _
     11: __________________________________________ _
     12: __________________________________________ _
     13: __________________________________________ _
     14: __________________________________________ _
     15: __________________________________________ _
                              PROPOSED TERMS:
 ______________________________      ______________________________
 ______________________________      ______________________________
 ______________________________      ______________________________
 ______________________________      ______________________________
 PF2=RESET; PF3=SIGNON; PF4=RELEASE; PF5=SELECTION; PF6=SUBQ
 PF10=ALPHA; PF11=HIERARCHY; PF12=STORE; PF13=CENTRAL SCREEN; PF20=TITLE/WNF
4B•                A                                  =-•PC LINE 6   COL 11
```