In *CADE-13 Workshop on Visual Reasoning*,
New Brunswick, NJ, July 1996.

Diagram-based Formalisms for the Verification of Reactive Systems*

Anca Browne, Luca de Alfaro, Zohar Manna,
Henny B. Sipma and Tomás E. Uribe
anca|luca|manna|sipma|uribe@cs.stanford.edu

Computer Science Department, Stanford University
Stanford, CA. 94305

Abstract

Graphical formalisms are an increasingly important component of our research on the specification and verification of reactive systems. We briefly describe diagram-based verification methods we have developed for verifying temporal properties of infinite-state reactive systems, as well as for the incremental analysis and refinement of systems and specifications.

1 Introduction

Reactive systems have an ongoing interaction with their environment. They include distributed and concurrent algorithms, hardware systems, and control programs. *Linear-time temporal logic* has proved to be a convenient language for expressing *safety*, *progress* and *response* properties of reactive systems [MP91, MP95]. We are interested in the formal verification of temporal properties of reactive systems by deductive and algorithmic means.

The classical approach for verifying temporal properties of reactive systems is based on *verification rules*, which reduce the system validity of a temporal property to the general validity of a set of first-order *verification conditions*. While this methodology is complete, relative to the underlying first-order reasoning, the proofs do not always reflect an intuitive understanding of the system and its specification; without this intuition, the proofs can be difficult to construct.

We address the need for a more intuitive approach to verification by using *diagram-based formalisms*. Our diagrams are graphs whose vertices are labeled with first-order formulas (*assertions*), representing sets of system states, and whose edges represent possible system transitions. In the following, we give an overview of three diagram-based formalisms for the verification of temporal properties of systems, which correspond to different methods of proof construction. *Verification diagrams* (Section 2) correspond to a completed, direct proof, and offer a compact representation of the necessary verification conditions. *Deductive model checking* (Section 3) uses the diagram representation to conduct an exhaustive search for a counterexample, giving a semi-automatic (but incomplete) proof procedure. *Fairness diagrams* (Section 4) represent abstractions of the system, used for the gradual construction of a proof by stepwise diagram transformations.

Some features shared by these formalisms are:

- Diagrams (or sequences of diagrams) are formal proof objects, which succinctly represent a set of verification conditions that replaces a combination of textual verification rules.

*This research was supported in part by the National Science Foundation under grant CCR-92-23226, the Advanced Research Projects Agency under NASA grant NAG2-892, the United States Air Force Office of Scientific Research under grant F49620-93-1-0139, the Department of the Army under grant DAAH04-95-1-0317, and a gift from Intel Corporation.

- The graphical nature of diagrams makes them easier to construct and understand than text-based proofs and specifications.
- Diagrams can describe and verify infinite-state systems using a finite and often compact representation.
- The construction of a diagram can be incremental, starting from a high-level outline and then filling in details as necessary. The diagrams for a given system can serve as documentation; they can also be re-used when similar proofs are carried out for similar systems, or when a system is refined.
- The verification conditions are *local* to the diagram; failed verification conditions point to missing edges or nodes, or possible bugs in the system. The necessary global properties of diagrams can be proved algorithmically.

For details on our verification framework, see [MP91, MP95]. Appendix A summarizes the definitions and notation used in the following presentation.

2 Verification Diagrams

Verification Diagrams, introduced by Manna and Pnueli [MP94], provide a graphical representation of the direct proof of temporal properties. A verification diagram is a directed graph, where nodes are labeled with assertions and edges are labeled with transitions. A single node can be selected as *terminal*, and should have not outgoing edges. Edges can be *single*, *double* or *solid*. Each transition can label at most one kind of edge leaving a given node.

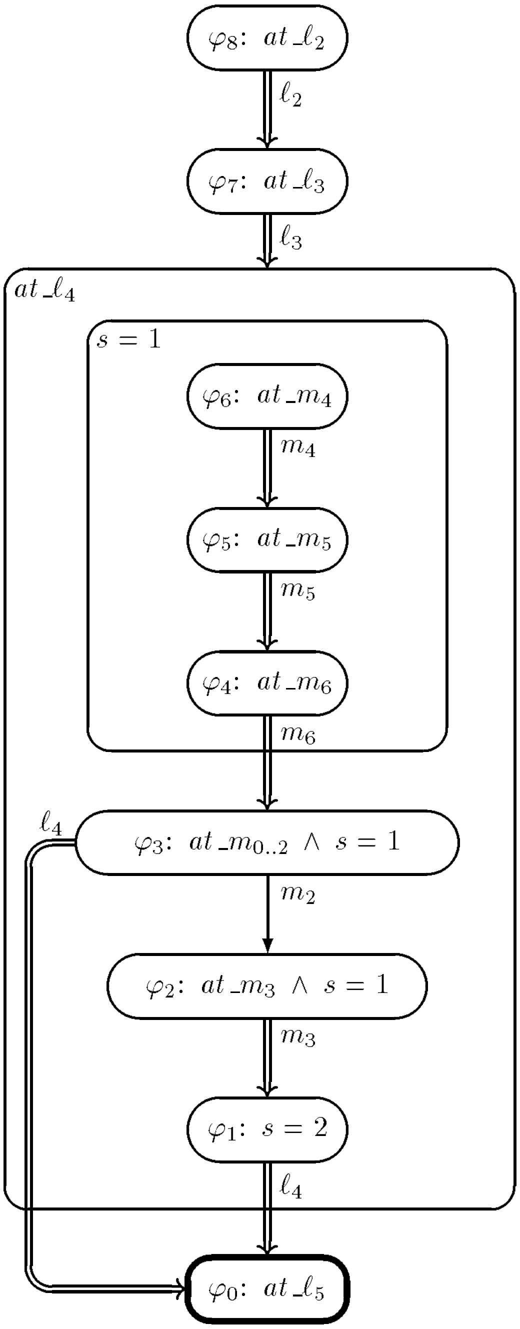
A set of verification conditions is associated with a diagram as follows: If transition τ labels a single edge departing a node labeled φ, reaching nodes labeled $\{\varphi_1, \ldots, \varphi_k\}$, the verification condition associated with φ and τ is $\{\varphi\}\,\tau\,\{\varphi \,\vee\, \varphi_1 \,\vee\, \ldots \,\vee\, \varphi_k\}$; for double or solid edges, which must be labeled by just and compassionate transitions respectively, the verification condition is $\{\varphi\}\,\tau\,\{\varphi_1 \,\vee\, \ldots \,\vee\, \varphi_k\}$. If τ does not label an outgoing edge from a node labeled φ, the verification condition $\{\varphi\}\,\tau\,\{\varphi\}$ is generated. If τ labels a double edge, the verification condition $\varphi \rightarrow \mathit{enabled}(\tau)$ is generated; for solid edges, the verification condition is $\Box(\varphi \rightarrow \Diamond(\neg\varphi \,\vee\, \mathit{enabled}(\tau)))$.

An *invariance* diagram is one with no terminal node; if its nodes are labeled by $\{\varphi_1, \ldots, \varphi_n\}$ and all of its associated verification conditions are valid, then the property

$$\Box\left((\bigvee_{j=1}^{m} \varphi_j) \rightarrow \Box(\bigvee_{j=1}^{m} \varphi_j) \right)$$

is valid over the given system $\mathcal{S}$. Other classes of diagrams can be used to prove general safety or response properties [MP94].

We have implemented verification diagrams as part of the STeP (Stanford Temporal Prover) system [BBC$^+$96], which includes an X-windows diagram editor. A *verification diagram rule* in STeP takes a temporal property φ and a diagram, checks that the diagram is well-formed with respect to φ, and generates the corresponding set of verification conditions as subgoals. In our experience, diagrams are the most convenient method available in STeP for the deductive verification of complex properties of large systems. For large diagrams, a hierarchical mechanism is provided, where given nodes can be bound to sub-diagrams that are separately specified.



To make diagrams more succinct, we use *encapsulation conventions*, based on those of Statecharts [Har87]. The assertion labeling a compound node is added, as a conjunct, to its subnodes. Edges leaving (entering) a compound node are interpreted as leaving (entering) all of its subnodes. These graphical abbreviations greatly increase the readability of diagrams, and make them easier to draw.

This CHAIN diagram proves that $\Box(at\_\ell_2 \rightarrow \Diamond\, at\_\ell_5)$ is valid for the MUX-PET program of Figure 1. φ_0 is the *terminal node*, with no outgoing edges. Double edges are be labeled with *helpful transitions*, which must be just. The diagram must be acyclic, and every non-terminal node should have a double edge departing from it (m_2 labels a *single* edge).
Transitions that do not label any outgoing edge must (if taken) stay within the given node. The helpful transitions must be taken since they are just and proved to be continuously enabled at the given nodes. (Compassion does not play a role in this example.) Thus, any computation that starts at node φ_8 must progress down the diagram until reaching φ_0. In STeP, this diagram generates 38 verification conditions. All are automatically proven to be valid, given automatically generated *local invariants* [BBM95].

CHAIN diagrams generalize to RANK diagrams, where an unbounded number of steps may be necessary to reach the terminal node. The acyclicity requirement is dropped, and replaced by *ranking functions*, which define a well-founded order that is always decreased by the helpful transitions.

Verification diagrams can be adapted to parameterized (N-component) designs [MP94], as well as real-time and hybrid systems [MP96].

Generalized and Modular Verification Diagrams: Similarly to verification rules, each class of Verification Diagrams in [MP94] is tailored to a particular class of temporal properties. *Generalized Verification Diagrams* [BMS95] extend the framework to arbitrary temporal formulas.

The validity of an arbitrary temporal formula φ for a given system $\mathcal{S}$ can be established by constructing an appropriate generalized verification diagram Ψ. A set of first-order verification conditions associated with the diagram Ψ establishes that the set of computations of $\mathcal{S}$, $\mathcal{L}(\mathcal{S})$, is a subset of $\mathcal{L}(\Psi)$, the computations represented by the diagram. This set is then shown to be a subset of $\mathcal{L}(\varphi)$, the sequences satisfying the temporal formula φ. Generalized verification diagrams offer a complete verification methodology: if an FTS $\mathcal{S}$ satisfies a formula φ then there is always a diagram Ψ such that $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\Psi) \subseteq \mathcal{L}(\varphi)$ [BMS95].

Modular Verification Diagrams [BMS96] allow the combination of several generalized verification diagrams into a single proof. The combined set represents the intersection of the languages described by each diagram. In this way, individual diagrams can be kept small, and diagrams can be re-used more conveniently. Modular verification diagrams can verify a class of properties equivalent to ω*-automata*, a strictly larger class than the properties that can be proved by regular verification diagrams or verification rules.

3 Deductive Model Checking

In a direct proof by verification diagram, the final, complete diagram must be constructed by the user. We have recently developed alternative, complementary verification formalisms where the proof is obtained incrementally through a set of transformations, combining deductive and algorithmic tools.

Given a linear-time temporal logic formula φ, the *formula tableau* for φ is a finite directed graph that represents all the models of φ (see e.g. [KMMP93]). *Model checking* verifies a system property φ by checking the emptiness of the *product graph* between the transition graph of $\mathcal{S}$ and the tableau for $\neg\varphi$. Since the state-space of the system needs to be generated explicitly, this approach is limited to finite-state systems only. (*Symbolic model checking* using BDDs [BCM+92] is similarly restricted to the finite-state case.)

We have generalized this classical model checking procedure to operate on an abstraction of the product graph, using first-order assertions to represent possibly infinite sets of states [SUM96]. This results in an interactive model checking procedure, where the user can choose the manner in which an abstraction of the state-space is explored. The resulting graphical representation is quite similar to verification diagrams; nodes are labeled by assertions, and edges are labeled by sets of transitions that can possibly move the system from one abstract state to the next. However, the goal now is to show that the corresponding set of computations is empty. If this is not the case, a counterexample can be produced, provided that appropriate formulas are proved valid.

This approach combines some of the advantages of deductive and algorithmic verification: the process is goal-directed and incremental, and can handle infinite-state systems. User input can direct the search for a counterexample and help rule out large portions of the search space before they are expanded to the state level, combating the *state-space explosion problem* common in model checking. The main graphical challenge is to present the abstracted state-space in a way where the user can easily choose what to do next.

4 Fairness Diagrams

Fairness diagrams [dAM96] are graphical abstractions of system behavior, and can combine the direct and indirect proof approaches above. The diagram represents the possible system states and transitions; the progress and response properties of the system are encoded by *fairness constraints*, which generalize the usual notions of fairness. Given a system and a temporal specification, a proof begins with an initial fairness diagram that directly corresponds to the system. This diagram is then transformed into one which corresponds directly to the specification, or which can be shown to satisfy it by purely algorithmic methods.

The transformations on fairness diagrams represent elementary system analysis steps. A first class of transformations relies on the construction of *simulation relations* between diagrams, and enables the study of the system's safety properties. A second class of transformations study progress and response properties, modifying or adding fairness constraints. Each transformation is justified by a set of first-order verification conditions that relate the two diagrams. The combined set of transformations is complete for proving general temporal properties of reactive systems (again, relative to first-order reasoning).

Diagram transformations allow a flexible style of proof construction. The system can be studied incrementally as well as modularly. Different system components can be studied separately and then expressed as sub-diagrams that represent their relevant properties. Note that while stepwise transformations facilitate the gradual construction of a proof, the proof itself is a more complex formal object when compared to a verification diagram, since it consists of a sequence of diagram transformations and their associated verification conditions.

As in standard deductive verification, all our formalisms can use previously proven properties and automatically generated invariants [BBM95] in the course of a proof. Furthermore, propagation and approximation mechanisms similar to those used in the *assertion graphs* of [BBM95] can be adapted to our diagrams as well, resulting in more automatic verification tools.

References

[BBC+96] N. Bjørner, A. Browne, E. Chang, M. Colón, A. Kapur, Z. Manna, H.B. Sipma, and T.E. Uribe. STeP: Deductive-algorithmic verification of reactive and real-time systems. In *Proc. 8^{th} Intl. Conference on Computer Aided Verification*. Springer-Verlag, July 1996.

[BBM95] N. Bjørner, A. Browne, and Z. Manna. Automatic generation of invariants and intermediate assertions. In 1^{st} *Intl. Conf. on Principles and Practice of Constraint Programming*, volume 976 of *LNCS*, pages 589–623. Springer-Verlag, September 1995.

[BCM+92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[BMS95] A. Browne, Z. Manna, and H.B. Sipma. Generalized verification diagrams. In *15th Conference on the Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *LNCS*, pages 484–498, December 1995.

[BMS96] A. Browne, Z. Manna, and H.B. Sipma. Modular verification diagrams. Technical report, Computer Science Department, Stanford University, 1996.

[dAM96] L. de Alfaro and Z. Manna. Temporal verification by diagram transformations. In *Proc. 8^{th} Intl. Conference on Computer Aided Verification*, July 1996.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Sci. Comp. Prog.*, 8:231–274, 1987.

[KMMP93] Y. Kesten, Z. Manna, H. McGuire, and A. Pnueli. A decision algorithm for full propositional temporal logic. In C. Courcoubetis, editor, *Proc. 5^{th} Intl. Conference on Computer Aided Verification*, volume 697 of *LNCS*, pages 97–109. Springer-Verlag, 1993.

[MP91] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1991.

[MP94] Z. Manna and A. Pnueli. Temporal verification diagrams. In *Proc. Int. Symp. on Theoretical Aspects of Computer Software*, volume 789 of *LNCS*, pages 726–765. Springer-Verlag, 1994.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.

[MP96] Z. Manna and A. Pnueli. Clocked transition systems. Technical Report STAN-CS-TR-96-1566, Department of Computer Science, Stanford University, April 1996.

[SUM96] H.B. Sipma, T.E. Uribe, and Z. Manna. Deductive model checking. In *Proc. 8^{th} Intl. Conference on Computer Aided Verification*. Springer-Verlag, July 1996.

A Background: Fair Transition Systems and Deductive Verification

Specifying Systems and Properties: We represent reactive systems as *fair transition systems*. A fair transition system is given by a set of variables $\mathcal{V}$, an *initial condition* Θ, and a set of *transitions* $\mathcal{T}$. A finite set of *system variables* $V \subset \mathcal{V}$ determines the possible states of the system. Θ is an assertion over the system variables. Each transition τ is described by its *transition relation* $\rho_\tau(\vec{x}, \vec{x}')$, an assertion over the system variables $\vec{x}$ and a set of *primed variables* $\vec{x}'$ indicating their next-state values. We define $\{\varphi\}\,\tau\,\{\psi\} \stackrel{\text{def}}{=} (\varphi(\vec{x}) \wedge \rho_\tau(\vec{x}, \vec{x}')) \rightarrow \psi(\vec{x'})$, and $enabled(\tau) \stackrel{\text{def}}{=} \exists \vec{x}'. \rho_\tau(\vec{x}, \vec{x}')$.

A computation of $\mathcal{S}$ is an infinite sequence of states $s_0, s_1, \ldots$, where s_0 must satisfy Θ and for every s_i there is a transition $\tau \in \mathcal{T}$ such that (s_i, s_{i+1}) satisfy ρ_τ (and we say that τ is *taken*). The transitions in $\mathcal{T}$ can be optionally designated as *just* or *compassionate*. Just (or *weakly fair*) transitions cannot be continuously enabled without ever being taken. Compassionate (or *strongly fair*) transitions cannot be enabled infinitely often without being taken.

$$
\begin{array}{c}
\textbf{local} \quad y_1, y_2 \; : \textbf{boolean where } y_1 = y_2 = \mathrm{F} \\
\qquad\quad s \; : [1..2] \\
P_1 :: \left[\begin{array}{l}
\ell_0\!: \textbf{while } \mathrm{T} \textbf{ do} \\
\quad \left[\begin{array}{l}
\ell_1\!: \textbf{noncritical} \\
\ell_2\!: y_1 := \mathrm{T} \\
\ell_3\!: s := 1 \\
\ell_4\!: \textbf{await } \neg y_2 \textbf{ or } s = 2 \\
\ell_5\!: \textbf{critical} \\
\ell_6\!: y_1 := \mathrm{F}
\end{array}\right]
\end{array}\right]
\; || \;
P_2 :: \left[\begin{array}{l}
m_0\!: \textbf{while } \mathrm{T} \textbf{ do} \\
\quad \left[\begin{array}{l}
m_1\!: \textbf{noncritical} \\
m_2\!: y_2 := \mathrm{T} \\
m_3\!: s := 2 \\
m_4\!: \textbf{await } \neg y_1 \textbf{ or } s = 1 \\
m_5\!: \textbf{critical} \\
m_6\!: y_2 := \mathrm{F}
\end{array}\right]
\end{array}\right]
\end{array}
$$

Figure 1: Peterson's algorithm for mutual exclusion (MUX-PET)

Figure 1 shows program MUX-PET, which implements Peterson's mutual exclusion algorithm, in the Simple Programming Language (SPL) of [MP95]. Such programs can be naturally translated into corresponding fair transition systems.[1] To each process corresponds a *control variable*. For MUX-PET, the control variables for P_1 and P_2 range over locations $\{\ell_0, \ldots, \ell_6\}$ and $\{m_0, \ldots, m_6\}$, Assertions at_m_i, at_ℓ_j indicate that control resides at i, j. All transitions are assumed to be just, except for those associated with the **noncritical** statements.

Specifications are expressed as formulas in *linear-time temporal logic*. *Assertions*, or state-formulas, are first-order formulas with no temporal operators, and can include quantifiers. Temporal formulas are constructed from assertions, boolean connectives, and the usual *future* ($\Box, \Diamond, \bigcirc, \mathcal{U}, \mathcal{W}$) and *past* (,, , $\mathcal{B}$, $\mathcal{S}$) temporal operators. For instance, the formula $\Box \neg(at\_\ell_5 \wedge at\_m_5)$ expresses mutual exclusion for MUX-PET.

Deductive Verification: *Verification rules* reduce the validity of a given temporal property over a given system S to the general validity of a set of first-order *verification conditions*. For example, the *general invariance rule*, which can prove a property of the form $\Box\, p$ for an assertion p, requires finding a strengthened assertion φ such that the following verification conditions are valid: (1) $\varphi \rightarrow p$, (2) $\Theta \rightarrow \varphi$ (that is, φ holds initially), and (3) $\{\varphi\}\,\tau\,\{\varphi\}$ for each transition $\tau \in S$ (that is, φ is preserved by all transitions). Other verification rules are available for proving different classes of temporal properties [MP95].

[1] The STeP system [BBC$^+$96] parses SPL programs into fair transition systems, or can take transition systems directly as input.