Xs Are for Trajectory Evaluation, Booleans Are for Theorem Proving

Mark D. Aagaard[1], Thomas F. Melham[2], and John W. O'Leary[1]

[1] Strategic CAD Labs, Intel Corporation, JFT-102
5200 NE Elam Young Parkway, Hillsboro, OR 97124, USA
{maagaard,joleary}@ichips.intel.com

[2] Department of Computing Science, University of Glasgow
Glasgow, Scotland, G12 8QQ
tfm@dcs.gla.ac.uk

Abstract. This paper describes a semantic connection between the symbolic trajectory evaluation model-checking algorithm and relational verification in higher-order logic. We prove a theorem that translates correctness results from trajectory evaluation over a four-valued lattice into a shallow embedding of temporal operators over Boolean streams. This translation connects the specialized world of trajectory evaluation to a general-purpose logic and provides the semantic basis for connecting additional decision procedures and model checkers.

1 Introduction

The well-known limits to BDD-based model-checking techniques have motivated a great deal of interest in combining model-checking with theorem proving [3,11,9,6]. The foundation of any such hybrid verification approach is a semantic connection between the logic of properties in the model checker and the logic of the theorem prover. Symbolic trajectory evaluation [16] is a highly effective model checker for datapath verification. It has been combined with theorem proving and the combination has been used effectively on complex industrial circuits [1,12]. However, two of the features that make trajectory evaluation so effective as a model checker create difficulties or limitations in the theorem proving domain. Trajectory evaluation's temporal logic has limited expressability and operates over a lattice of values containing notions of contradiction ($\top$) and unknown (X).

In this paper, we formally verify a semantic link from symbolic trajectory evaluation to higher-order logic. This link allows trajectory evaluation to be used as a decision procedure without encumbering the theorem proving world with the complications and limitations of trajectory evaluation. The trajectory evaluation temporal operators are defined in a shallow embedding of predicates over streams and the lattice domain is converted to simple Booleans. This trans-

© Springer-Verlag Berlin Heidelberg 1999

lates trajectory evaluation results into the conventional "relations-over-Boolean-streams" approach to hardware modeling in higher-order logic [7].[1]

We believe that the relational world is the right target domain for connecting model checking engines. Each model checking algorithm typically has its own temporal logic. By translating results into higher-order logic, the vestiges of the individual model checkers are removed, allowing the full power of general-purpose theorem proving to be brought to bear.

To give some intuition about the two worlds we reason about and the connection between them, consider the NAND-DELAY circuit in Figure 1. Figure 2 presents simple correctness statements for both the trajectory evaluation and relational styles. We will use this circuit as a running example throughout this paper. In this example, for simplicity, we consider the NAND gate to be zero delay. Trajectory evaluation typically uses a more detailed timing model of circuits.

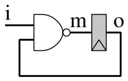


Fig. 1. Simple example circuit NAND_DELAY

$$i \overset{def}{=} \texttt{"}i\texttt{"}$$
$$o \overset{def}{=} \texttt{"}o\texttt{"}$$
$$\models_{ckt} \left[\begin{array}{l}(i \text{ is } b_1)\text{and} \\ (o \text{ is } b_2)\end{array} \Longrightarrow\!\!\!> (\mathsf{N}\ (o \text{ is } \neg(b_1 \wedge b_2))\right]$$

Trajectory-evaluation verification

$$\begin{array}{l} ckt_{\texttt{HOL}}\ i\ o \overset{def}{=} \\ \quad \exists m. \\ \qquad (\forall t.\ m\ t \quad = \texttt{NAND}(i\ t)(o\ t)) \ \wedge \\ \qquad (\forall t.\ o\ (t+1) = m\ t) \end{array}$$

$$\begin{array}{l} spec_{\texttt{HOL}}\ i\ o \overset{def}{=} \\ \quad \forall t. o\ (t+1) = \texttt{NAND}(i\ t)(o\ t) \end{array}$$

$$\forall i, o.\ ckt_{\texttt{HOL}}\ i\ o \implies spec_{\texttt{HOL}} i\ o$$

Relational style verification

Fig. 2. Example stream and trajectory evaluation verifications

Trajectory evaluation is based on symbolic simulation. Correctness statements are of the form $\models_{ckt} [ant \Longrightarrow\!\!\!> cons]$, where $\Longrightarrow\!\!\!>$ is similar, but not identical to, implication (details are given in Section 2). The *antecedent ant* gives an initial state and input stimuli to the circuit *ckt*, while the *consequent cons* specifies the desired response of the circuit. Circuits are black boxes—their implementations are not user visible. Circuit *nodes* are named by strings. In the example, the antecedent drives the nodes "*i*" and "*o*" with the Boolean variables b_1 and b_2 at the initial step of the verification. The consequent says that at the next time step the node "*o*" has the value $\neg(b_1 \wedge b_2)$.

[1] In the rest of the paper we will take the phrase "relational" style to mean "relations over Boolean streams" with a shallow embedding of temporal operators as predicates.

In relational verification, correctness statements are of the form $\mathit{ckt}\ i\ o \Longrightarrow \mathit{spec}\ i\ o$, where $\Longrightarrow$ is true implication. Signals (e.g., i, m, and o) are modelled by *streams*, which are functions from time to values. Both the circuit and the specification are relations over these infinite streams. A stream satisfies a circuit if it is a series of values that could be observed on the corresponding signals in the circuit. The correctness criterion for the example says that if the streams i and o satisfy the circuit model, then they must conform to the specification.

At a very cursory level, a mapping from the trajectory evaluation result in Figure 2 to the relational world would result in:

$$\begin{array}{l} \forall i, o.\ \forall t. \\ \quad ((i\ t) = b_1) \land ((o\ t) = b_2) \land (\mathit{ckt}\ i\ o) \\ \quad \Longrightarrow \\ \quad (o\ (t+1)) = \neg(b_1 \land b_2) \end{array}$$

Substituting for b_1 and b_2 throughout the expression gives:

$$\forall i, o.\ \forall t.\ \mathit{ckt}\ i\ o \implies (o\ (t+1)) = \neg((i\ t) \land (o\ t))$$

Technically, this description is not quite correct, but it gives the intuition behind our result that correctness statements in trajectory evaluation imply relational correctness statements. Section 5.2 shows the actual process and results for the NAND-DELAY circuit. The difficulties arise in details such as translation between different semantic domains (e.g., Boolean and lattice valued streams) and the treatment of free variables in trajectory formulas.

Our main result is a formal translation from trajectory evaluation's temporal operators over lattices to a shallow embedding of the temporal operators over Boolean streams. We prove that any result verified by the trajectory evaluation algorithm will hold in the relational world. This allows trajectory evaluation to be used as a decision procedure in a theorem prover without changing the relational style of verification used in the theorem prover.

It is interesting to note that our result is an implication, not an "if-and-only-if"; that is, we do not guarantee that every statement provable in the relational world will also hold in trajectory evaluation. The problem stems from the differences in how the relational world and the trajectory evaluation world handle contradictions in the circuit and antecedent. Joyce and Seger gave an extra constraint on trajectory evaluation that can be used to prove an if-and-only-if relationship [10]. The constraint requires reasoning about contradictions and top values. Because we use trajectory evaluation as a decision procedure, we are able to avoid the burden of reasoning about contradictions.

1.1 Organization of the Paper

Figure 3 is a roadmap of the paper. We begin with a presentation of trajectory assertions (the specifications for symbolic trajectory evaluation) over the standard four-valued lattice (Section 2). Our verification relies on two major steps:

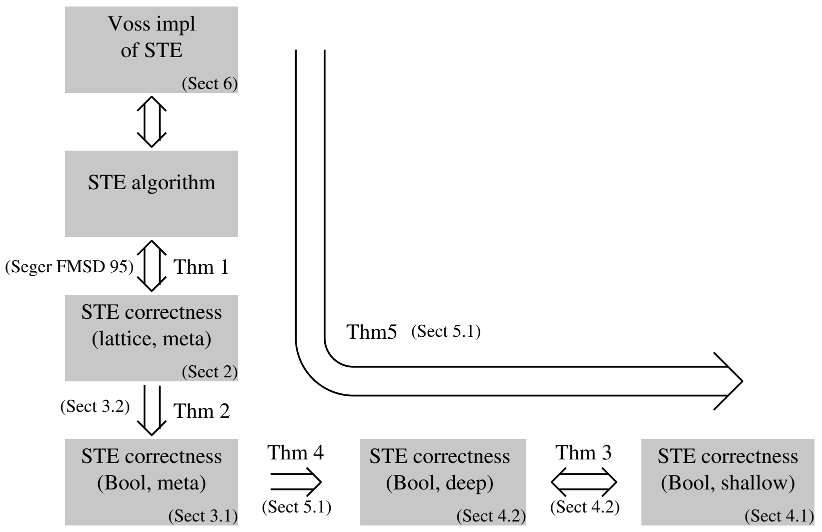


Fig. 3. Roadmap of the paper

from the four-valued lattice to Booleans and from a deep embedding of the temporal operators to a shallow embedding. In Section 3 we introduce our definition of trajectory assertions over Booleans and prove that a trajectory assertion over lattice-valued streams implies the same result over Boolean-valued streams. We then prove that a shallow embedding of trajectory formulas as relations over streams is equivalent to a deep embedding (Section 4).

Our proof relating trajectory assertions over lattices to trajectory assertions over Booleans is a meta-logical proof about the semantics of the two languages and links the free (Boolean) variables that appear in these assertions. The conversion from the deep to shallow embedding is done in higher-order logic. In Section 5 we connect connect all of the pieces together to prove our final result.

1.2 Related Work

An early experiment in combining model checking and theorem proving was the HOL-Voss System [10]. Within HOL [7], Joyce and Seger proved the correspondence between a deep embedding of the Voss [8] "5-tuple" implementation of trajectory assertions (see Section 6) and a simple specification language that was deeply embedded in HOL. This allowed them to use Voss as an external decision procedure for HOL. Our work focuses on the logical content of trajectory assertions, independent of any particular implementation (e.g., Voss), and connects this to a shallow embedding of temporal operators in higher-order logic.

The modal mu-calculus has been embedded in HOL [2] and in PVS [15]. In the PVS work, Rajan *et al* implemented a shallow embedding upon which they defined the $\forall$CTL* temporal logic. They connected a mu-calculus model checker and verified a number of abstraction opertations. It should be possible to translate a subset of the $\forall$CTL* formulas into predicates over Boolean streams, but given the complexity of $\forall$CTL*, it is difficult to estimate the feasibility of this approach.

Chou has given a set-theoretic semantics of trajectory evaluation, focusing on extensions to the basic algorithm [5]. Our work translates trajectory evaluation results into a form that can be seamlessly integrated with current practice in higher-order-logic theorem proving. It would be interesting to explore connections between the work presented here and Chou's, to find a seamless connection from extended trajectory evaluation to theorem-proving in higher-order logic.

2 Symbolic Trajectory Evaluation

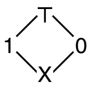
This section presents the logic of trajectory assertions. Our results are meta-logical, but for convenience we use a **bold** face logical-style notation to state our results. Our presentation of trajectory evaluation is comprised of three parts. After a few preliminary definitions; we proceed with Section 2.1, which describes the four-valued lattice. Section 2.2 overviews the circuit model used in trajectory evaluation. Finally, Section 2.3 describes the specification logic of trajectory evaluation.

First, some preliminaries. We suppose there is a set of *nodes*, naming observable points in circuits. A *stream* is a function from natural numbers representing time to data values in the stream. A *sequence* takes a node and returns the stream for that node. A *state* is a mapping from *nodes* to values. We typically use σ for sequences and s for states. Two convenient sequence operations are taking the suffix and transposing a sequence so that it is a stream of states.

$$\begin{array}{lll}
\alpha\ \texttt{stream} & \overset{type}{=} & \mathcal{N} \to \alpha \\
\alpha\ \texttt{sequence} & \overset{type}{=} & \texttt{node} \to \alpha\ \texttt{stream} \\
\alpha\ \texttt{state} & \overset{type}{=} & \texttt{node} \to \alpha \\
\text{suffix:} \quad \sigma_i & \overset{def}{=} & \lambda n.\ \lambda t.\ (\sigma\ n\ (t+i)) \\
\text{transpose:} \quad \sigma^{\mathsf{T}} & \overset{def}{=} & \lambda t.\ \lambda n.\ (\sigma\ n\ t) \\
\sigma^{\mathsf{T}} & :: & (\alpha\ \texttt{state})\ \texttt{stream}
\end{array}$$

2.1 The Four Valued Lattice

In this paper, the only lattice that we use is the four-valued lattice shown below. The theory of trajectory evaluation works equally well over all complete lattices. However this lattice simplifies the presentation and is the lattice used by the Voss implementation of trajectory evaluation [8]. Our mathematical development also is based on this lattice—generalizing our results would be of theoretical interest, but not of immediate practical benefit to us.



The ordering over lattice values shown above defines the ordering relation $\sqsubseteq$, which we lift pointwise and overload over streams and states. We inject the set of Boolean values to lattice values with the postfix operator $\downarrow$ (read "drop", Definition 1), which maps the Boolean values `T` and `F` to their counterparts in the lattice. Drop is lifted pointwise to states, to sequences, and to state streams.

Definition 1. *Dropping from Boolean to lattice values*

$$\texttt{F}\!\downarrow \overset{def}{=} \mathsf{0}$$
$$\texttt{T}\!\downarrow \overset{def}{=} \mathsf{1}$$

2.2 Circuit Models

In our description of circuit models, we will refer to Table 1, which gives the lattice transition function for the example NAND-DELAY circuit. In trajectory evaluation, the circuit model is given by a next state function $\mathbf{Y}$ that takes a circuit and maps states to states:

$$\mathbf{Y} \;::\; \texttt{ckt} \rightarrow \texttt{lattice state} \rightarrow \texttt{lattice state}$$

A lattice state is an assignment to circuit nodes of values drawn from the four valued lattice. The first argument to $\mathbf{Y}$ identifies the particular circuit of interest, and for the present purposes may be regarded as an uninterpreted constant. Intuitively, the next state function expresses a constraint on the set of possible states into which the circuit may go for any given state. Suppose the circuit is in state s, then $\mathbf{Y}(s)$ will give the least specified state the system can transition to. Here, "least specified" means that if a node can take on both 1 and 0 values in the next state, then $\mathbf{Y}(s)$ will assign the value X to that node.

Table 1. Lattice transition function for NAND-DELAY circuit

$\langle\ i\ \ o\ \rangle$	$\langle\ i'\ o'\ \rangle$	
0 0	X 1	
0 1	X 1	
0 X	X 1	$\longleftarrow$ o can initially be X, and o' is still defined
1 0	X 1	
1 1	X 0	
1 X	X X	$\longleftarrow$ o' is unknown, because o is unknown
X 0	X 1	$\longleftarrow$ i can initially be X, and o' is still defined
X 1	X X	$\longleftarrow$ o' is unknown, because i is unknown
X X	X X	

A critical requirement for trajectory evaluation is that the next-state function be monotonic, which is captured in Axiom 1.

Axiom 1. *Monotonicity of* $\mathbf{Y}$

$$\textbf{For all } s, s'.\ (s \sqsubseteq s') \textbf{ implies } (\mathbf{Y} s \sqsubseteq \mathbf{Y} s')$$

Monotonicity can be seen in the NAND-DELAY circuit by comparing a transition in which one of the current state variables (e.g., o) is X with a transition in which o is either 0 or 1. A bit of the algorithmic efficiency of trajectory evaluation is illustrated here. The initial value for some of the circuit nodes can be X and a meaningful result can still be verified. In this way, the lattice often allows trajectory evaluation to prove results with fewer BDD variables than would otherwise be needed.

A sequence σ is said to be *in the language* of a circuit (Definition 2) if the set of behaviors that the sequence encodes is a subset of the behaviors that the circuit can exhibit. This means that the result of applying $\mathbf{Y}$ to any element of the state stream σ^{T} is no more specified (with respect to the $\sqsubseteq$ ordering) than the succeeding element of σ^{T}.

Definition 2. *Sequence is in the language of a circuit*

$$\sigma \in \mathcal{L}\ ckt \overset{def}{=} \textbf{For all } t \geq 0.\ (\mathbf{Y}\ ckt\ (\sigma^{\mathsf{T}}\ t)) \sqsubseteq (\sigma^{\mathsf{T}}\ (t+1))$$

2.3 Trajectory Evaluation Logic

Trajectory evaluation correctness statements (known as *trajectory assertions*) are written as:

$$\models_{ckt} [ant \Longrightarrow cons]$$

where *ant* and *cons* are *trajectory formulas.* The intuition is that the antecedent *ant* provides stimuli to nodes in the circuit and the consequent *cons* specifies the values expected on nodes in the circuit. Before further describing trajectory assertions, we define trajectory formulas (Definition 3) and what it means for a sequence to satisfy a trajectory formula (Definition 4).

Definition 3. *Trajectory formulas*

$$\begin{array}{lll} f \overset{def}{=} & n \textsf{ is } 0 & \textit{// n has value 0} \\ \mid & n \textsf{ is } 1 & \textit{// n has value 1} \\ \mid & f_1 \textsf{ and } f_2 & \textit{// conjunction of formulas} \\ \mid & f \textsf{ when } g & \textit{// f is asserted only when g is true} \\ \mid & \textsf{N}\ f & \textit{// f holds in the next time step} \end{array}$$

where f, f_1, f_2 range over formulas; n ranges over the node names of the circuit; and g is a Boolean expression, commonly called a guard.

Trajectory formulas are guarded expressions defining values on nodes in the sequence. Guards may contain free variables. In fact, guards are the only place that free variables are allowed in the primitive definition of trajectory formulas. Syntactic sugar for is is commonly defined to allow guards in the value field as well. This is illustrated in Figure 1 and Section 5.2.

Definition 4 describes when a sequence σ satisfies a trajectory formula f. Satisfaction is defined with respect to an assignment ϕ of Boolean values to the variables that appear in the guards of the formula.

Definition 4. *Sequence satisfies a trajectory formula*

$$\begin{array}{lcl}
(\phi, \sigma) \models_{\mathsf{STE}} (n \;\mathsf{is}\; 0) & \overset{def}{=} & \sigma\ n\ 0 \sqsupseteq 0 \\
(\phi, \sigma) \models_{\mathsf{STE}} (n \;\mathsf{is}\; 1) & \overset{def}{=} & \sigma\ n\ 0 \sqsupseteq 1 \\
(\phi, \sigma) \models_{\mathsf{STE}} (f_1 \;\mathsf{and}\; f_2) & \overset{def}{=} & ((\phi, \sigma) \models_{\mathsf{STE}} f_1) \;\textbf{and}\; ((\phi, \sigma) \models_{\mathsf{STE}} f_2) \\
(\phi, \sigma) \models_{\mathsf{STE}} (f \;\mathsf{when}\; g) & \overset{def}{=} & (\phi \models g) \;\textbf{implies}\; ((\phi, \sigma) \models_{\mathsf{STE}} f) \\
(\phi, \sigma) \models_{\mathsf{STE}} (\mathsf{N} f) & \overset{def}{=} & (\phi, \sigma_1) \models_{\mathsf{STE}} f
\end{array}$$

Where $\phi \models g$ means that the assignment that ϕ makes to the free variables in g renders g true.

We now have sufficient notation to define a *trajectory assertion* (Definition 5). In trajectory evaluation, correctness criteria are formulated as trajectory assertions.

Definition 5. *Trajectory assertion*

$$\phi \models_{ckt} [ant \Longrightarrow cons] \overset{def}{=}$$
$$\textbf{For all}\ \sigma.\ (\sigma \in \mathcal{L}\ ckt)\ \textbf{implies}\ ((\phi, \sigma) \models_{\mathsf{STE}} ant)\ \textbf{implies}\ ((\phi, \sigma) \models_{\mathsf{STE}} cons)$$

The fundamental theorem of trajectory evaluation [16] says the trajectory evaluation algorithm ($\texttt{STE}\ ckt\ ant\ cons$) computes the Boolean condition e on the free variables in ant and $cons$ if and only if any assignment ϕ satisfying e also proves the trajectory assertion $\phi \models_{ckt} [ant \Longrightarrow cons]$ (Theorem 1).

Theorem 1. *Correctness of STE algorithm*
For all circuits ckt, antecedents ant, and consequences cons, the implementation of the STE algorithm returns the value e (that is, $e = \texttt{STE}\ ckt\ ant\ cons$) if and only if:

$$\textbf{For all}\ \phi.\ \phi \models e\ \textbf{implies}\ \phi \models_{ckt} [ant \Longrightarrow cons]$$

3 Trajectory Logic over Boolean Streams

In this section we give a definition of trajectory logic over Boolean streams (as opposed to the standard lattice-valued streams in Section 2) and prove that trajectory evaluation results that hold over the four valued lattice also hold over Boolean streams. Boolean identifiers (e.g., next-state relations, sequences, and languages) will be distinguished from their lattice valued counterparts by marking them with a $\circ$, as in $\mathring{\mathbf{Y}}$, $\mathring{\sigma}$, and $\mathring{\mathcal{L}}$.

3.1 Definitions and Axioms

In the Boolean world, circuit behavior is modeled as a *relation* between current and next states. In contrast, circuit behavior in trajectory evaluation is modeled as a next state *function*. The Boolean next state relation, denoted $\mathring{\mathbf{Y}}$, has the type:

$$\mathring{\mathbf{Y}}\quad :: \texttt{ckt} \rightarrow \texttt{bool state} \rightarrow \texttt{bool state} \rightarrow \texttt{bool}$$

We write $\overset{\circ}{\mathbf{Y}}\ ckt$ as an infix operator, as in: $s(\overset{\circ}{\mathbf{Y}}\ ckt)s'$.

As a concrete example, the next state relation for the NAND-DELAY circuit of Figure 1 is defined by Table 2, where the vectors $\langle i, o\rangle$ and $\langle i', o'\rangle$ denote the current and next states of the input and output. Note that the non-determinism that was represented by Xs in $\mathbf{Y}$ (Table 1) appears as multiple next states with the same current state in Table 2.

$\langle\ i$	$,\ o\ \rangle$	$\langle\ i'$	$,\ o'\ \rangle$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	0	1

Table 2. Boolean next-state relation for NAND-DELAY

Given a circuit's next state relation $\overset{\circ}{\mathbf{Y}}$, we say that a Boolean sequence $\overset{\circ}{\sigma}$ is *in the language* of the circuit when consecutive states in the state stream $\overset{\circ}{\sigma}^{\mathsf{T}}$ are included in the next-state relation (Definition 6).

Definition 6. *Boolean sequence is in the language of a circuit:*

$$\overset{\circ}{\sigma} \in \overset{\circ}{\mathcal{L}}\ ckt \overset{def}{=} \textbf{For all } t \geq 0.\ (\overset{\circ}{\sigma}^{\mathsf{T}} t)\ (\overset{\circ}{\mathbf{Y}}\ ckt)\ (\overset{\circ}{\sigma}^{\mathsf{T}}(t+1))$$

We now define when a Boolean sequence $\overset{\circ}{\sigma}$ satisfies a trajectory formula f (Definition 7). The only distinction between $\overset{\circ}{\models}$ and satisfaction over lattice sequences ($\models_{\mathsf{STE}}$) is that for the formulas $(n \textsf{ is } 0)$ and $(n \textsf{ is } 1)$, satisfaction is defined in terms of values in the Boolean domain rather than the lattice domain.

Definition 7. *Boolean sequence satisfies a trajectory formula*

$$\begin{array}{lll}
(\phi, \overset{\circ}{\sigma}) \overset{\circ}{\models} (n \textsf{ is } 0) & \overset{def}{=} & (\overset{\circ}{\sigma}\ n\ 0) = \texttt{F} \\
(\phi, \overset{\circ}{\sigma}) \overset{\circ}{\models} (n \textsf{ is } 1) & \overset{def}{=} & (\overset{\circ}{\sigma}\ n\ 0) = \texttt{T} \\
(\phi, \overset{\circ}{\sigma}) \overset{\circ}{\models} (f_1 \textsf{ and } f_2) & \overset{def}{=} & ((\phi, \overset{\circ}{\sigma}) \overset{\circ}{\models} f_1)\ \ \textbf{and}\ \ ((\phi, \overset{\circ}{\sigma}) \overset{\circ}{\models} f_2) \\
(\phi, \overset{\circ}{\sigma}) \overset{\circ}{\models} (f \textsf{ when }\ g) & \overset{def}{=} & (\phi \models g)\ \textbf{implies}\ ((\phi, \overset{\circ}{\sigma}) \overset{\circ}{\models} f) \\
(\phi, \overset{\circ}{\sigma}) \overset{\circ}{\models} (\textsf{N}\ f) & \overset{def}{=} & (\phi, \overset{\circ}{\sigma}_1) \overset{\circ}{\models} f
\end{array}$$

3.2 Correctness of Boolean Valued Trajectory Evaluation

To link the worlds of lattice and Boolean based trajectory evaluation, we require that the two models of the circuit behavior ($\mathbf{Y}$ and $\overset{\circ}{\mathbf{Y}}$) describe the same behavior. Axiom 2 says that if two Boolean states s and s' satisfy the next-state relation $\mathring{\mathbf{Y}}$, then the result of applying the next-state function $\mathbf{Y}$ to the dropped versions of s results in a state that is no higher in the lattice than $s'{\downarrow}$ ($\mathbf{Y}$ is a ternary extension of $\mathring{\mathbf{Y}}$).

Axiom 2. *Relating next-state relation and next-state function*

For all $ckt,\ s,\ s'.\ s\ (\mathring{\mathbf{Y}}\ ckt)\ s'$
implies
$(\mathbf{Y}\ ckt\ (s{\downarrow})) \sqsubseteq (s'{\downarrow})$

$$\begin{tikzcd}
s \arrow[rr, "\mathring{\mathbf{Y}}"] \arrow[d, "\downarrow"'] & & s' \arrow[d, "\downarrow"] \\
s{\downarrow} \arrow[r, "\mathbf{Y}"'] & \mathbf{Y}\, s{\downarrow} \arrow[r, phantom, "\sqsubseteq"] & s'{\downarrow}
\end{tikzcd}$$

Fig. 4. Illustration of Axiom 2

Axiom 2, which is illustrated in Figure 4, says that any Boolean next state s' possible in the relational model is consistent with the next state in the lattice-valued model. It also constrains $\mathring{\mathbf{Y}}\ ckt$ to return F whenever it is applied to two states s, s' that are inconsistent with $\mathbf{Y}\,ckt$ (e.g., unreachable states). Inconsistency is manifested by $\mathbf{Y}\,ckt\ s$ returning the lattice value $\top$ (top).

Theorem 2 makes the connection between trajectory assertions over lattice "values" and Boolean trajectory assertions. If a trajectory assertion holds over lattice-valued streams, then the same antecedent leads to the same consequent over Boolean-valued streams. This is the crux of connection from the lattice world to the Boolean world.

Theorem 2. *Translate trajectory logic from lattice to Boolean sequences.*

For all $ckt, ant, cons$.
  For all ϕ, σ.
    $\sigma \in \mathcal{L}\ ckt$ **implies**
      For all $t \geq 0$. $((\phi, \sigma_t) \models_{\mathsf{STE}} ant)$ **implies** $((\phi, \sigma_t) \models_{\mathsf{STE}} cons)$
  implies
  For all $\phi, \mathring{\sigma}$.
    $\mathring{\sigma} \in \mathring{\mathcal{L}}\ ckt$ **implies**
      For all $t \geq 0$. $\big((\phi, \mathring{\sigma}_t) \overset{\circ}{\models} ant\big)$ **implies** $\big((\phi, \mathring{\sigma}_t) \overset{\circ}{\models} cons\big)$

The proof of Theorem 2 relies on Lemmas 1 and 2. Lemma 1 says that if a Boolean sequence $\mathring{\sigma}$ is in the Boolean language of a circuit then the dropped version $\mathring{\sigma}{\downarrow}$ is in the lattice-valued language of the circuit. The proof of Lemma 1 is done by unfolding the definition of $\in \mathring{\mathcal{L}}$ and using Axiom 2.

Lemma 1. *Relationship between Boolean and ternary sequences*

$$\textbf{For all } \mathring{\sigma}.\ \mathring{\sigma} \in \mathring{\mathcal{L}}\ ckt\ \ \textbf{implies}\ \ \mathring{\sigma}{\downarrow} \in \mathcal{L}\ ckt$$

Lemma 2 relates satisfaction over Boolean sequences to satisfaction over lattice-valued sequences. Its proof is by induction over the structure of trajectory formulas, unfolding the definitions of $\overset{\circ}{\models}$ and $\models_{\mathsf{STE}}$, and employing properties of the drop operator (Lemma 3).

Lemma 2. *Satisfaction over Boolean and ternary sequences*

$$\textbf{For all } \phi, f, \mathring{\sigma}\,.\ (\phi, \mathring{\sigma}) \models^{\circ} f \ \textbf{ iff } \ (\phi, \mathring{\sigma}\!\downarrow) \models_{\mathsf{STE}} f$$

Lemma 3 says that the value of an element of a Boolean sequence $\mathring{\sigma}$ is $\mathtt{F}$ ($\mathtt{T}$) if-and-only-if the value of the dropped sequence at that point is higher in the lattice than $\mathsf{0}$ ($\mathsf{1}$). The lemma holds because $\mathtt{F}\!\downarrow = \mathsf{0}$, $\mathsf{0} \sqsupseteq \mathsf{0}$ (similarly, $\mathtt{T}\!\downarrow = \mathsf{1}$, and $\mathsf{1} \sqsupseteq \mathsf{1}$).

Lemma 3. *Properties of drop*

$$\textbf{For all } \mathring{\sigma}\,.\ (\mathring{\sigma}\ n\ 0 \ = \ \mathtt{F}) \ \textbf{ iff } \ ((\mathring{\sigma}\!\downarrow)\ n\ 0 \ \sqsupseteq \mathsf{0})$$
$$\textbf{For all } \mathring{\sigma}\,.\ (\mathring{\sigma}\ n\ 0 \ = \ \mathtt{T}) \ \textbf{ iff } \ ((\mathring{\sigma}\!\downarrow)\ n\ 0 \ \sqsupseteq \mathsf{1})$$

Theorem 2 is an implication and not an if-and-only-if result. The reason stems from Lemma 3, which causes Lemma 2 to be universally quantified over Boolean sequences, as opposed to lattice-valued sequences. Examining the case in which the trajectory formula f contains both n is $\mathsf{1}$ and n is $\mathsf{0}$ illustrates why Lemma 2 does not hold for all lattice-valued sequences. There are no Boolean sequences that satisfy both n is $\mathsf{1}$ and n is $\mathsf{0}$, but a lattice valued sequence in which $\sigma\ n\ 0 = \top$ would satisfy the formula.

4 Trajectory Logic as Relations over Streams

This section begins with a description of a shallow embedding of trajectory assertions in a higher-order logic[2] version of Boolean sequences. In the shallow embedding, trajectory formulas are predicates over Boolean sequences. In Section 4.2, we link the shallow embedding with a deep embedding of trajectory formulas. The deep embedding definitions mirror those in the metalogic from Section 3, and so we do not include them. Later, in Section 5 we use the deep embedding as an intermediate representation to connect the shallow embedding to trajectory assertions over Boolean streams from Section 3.

The decision to develop our shallow embedding via an intermediate deep embedding was made consciously. While we much prefer the shallow embedding for reasoning about properties of circuits stated as trajectory assertions, the deep embedding enables reasoning about trajectory logic itself (in particular, it allows quantification over trajectory formulas). We consider both activities important.

4.1 Definitions and Axioms

Circuits are implemented as relations over streams in our shallow embedding. Checking that a sequence is in the language of a circuit is done by simply applying the circuit relation to the sequence. We axiomatize the relationship between the language of circuits in our shallow embedding, the language of circuits in our deep embedding (`in_lang`), and the language of circuits over Boolean sequences

[2] This is not a mechanized implementation, but rather a paper description that could be implemented in a higher-order logic proof system.

($\in \mathring{\mathcal{L}}$) in Axiom 3. This is an axiom, rather than a lemma, because in this paper we do not give interpretations of circuits. Indeed, for much practical work we only require the ability to distinguish verification results obtained on different circuits, and for this purpose it is sufficient to leave circuits uninterpreted. A complete implementation of the work described here would need to prove that the implementation of circuits satisfies Axiom 3.

Axiom 3. *Relating languages in deep and shallow embeddings.*
A sequence $\mathring{\sigma}$ is in the language of a deeply-embedded circuit ckt ($\mathring{\sigma}$ $\underline{\texttt{in\_lang}}$ ckt) if and only if $\mathring{\sigma} \in \mathring{\mathcal{L}}\, ckt$.
A sequence $\mathring{\sigma}$ is in the language of a shallowly-embedded circuit ckt (ckt $\mathring{\sigma}$) if and only if $\mathring{\sigma}$ $\underline{\texttt{in\_lang}}$ ckt.

Definition 8 presents the trajectory formula type and the trajectory formula constructs `is`, `and`, `when`, and `N` for the shallow embedding in a higher-order logic. In the shallow embedding, a sequence satisfies a formula if applying the formula to the sequence yields true.

Definition 8. *Shallow embedding of trajectory formulas in Boolean streams*

$$\texttt{traj\_form} \overset{type}{=} (\texttt{bool traj}) \to \texttt{bool}$$

$$\begin{array}{lcl}
n \,\texttt{is}\, 1 & \overset{def}{=} & \lambda\mathring{\sigma}.\ \mathring{\sigma}\ n\ 0 = 1 \\
n \,\texttt{is}\, 0 & \overset{def}{=} & \lambda\mathring{\sigma}.\ \mathring{\sigma}\ n\ 0 = 0 \\
f_1 \,\texttt{and}\, f_2 & \overset{def}{=} & \lambda\mathring{\sigma}.\ (f_1\ \mathring{\sigma}) \wedge (f_2\ \mathring{\sigma}) \\
f \,\texttt{when}\, g & \overset{def}{=} & \lambda\mathring{\sigma}.\ g \Longrightarrow (f\ \mathring{\sigma}) \\
\texttt{N}\ f & \overset{def}{=} & \lambda\mathring{\sigma}.\ f\,\mathring{\sigma}_1
\end{array}$$

4.2 Verification of Shallow Embedding Against Deep Embedding

As mentioned previously, the deep embedding of trajectory formulas and satisfaction ($\underline{\texttt{is}}$, $\underline{\texttt{and}}$, $\underline{\texttt{when}}$, $\underline{\texttt{N}}$ and $\underline{\texttt{sat}}$) is not shown because it is a direct implementation of the metalogical presentation in Section 3. We identify the deeply embedded operators by underlining them.

Definition 9 defines a translation $[\![\cdot]\!]$ from deeply-embedded to shallowly-embedded trajectory formulas.

Definition 9. *Translation from deep to shallow embedding*

$$\begin{array}{lcl}
[\![(n \,\underline{\texttt{is}}\, 1)]\!] & \overset{def}{=} & n \,\texttt{is}\, 1 \\
[\![(n \,\underline{\texttt{is}}\, 0)]\!] & \overset{def}{=} & n \,\texttt{is}\, 0 \\
[\![(f_1 \,\underline{\texttt{and}}\, f_2)]\!] & \overset{def}{=} & [\![f_1]\!] \,\texttt{and}\, [\![f_2]\!] \\
[\![(f \,\underline{\texttt{when}}\, g)]\!] & \overset{def}{=} & [\![f]\!] \,\texttt{when}\, g \\
[\![(\underline{\texttt{N}}\ f)]\!] & \overset{def}{=} & \texttt{N}\ [\![f]\!]
\end{array}$$

The core of the relationship between trajectory formulas in the deep and shallow embeddings is captured in Theorem 3. The theorem says that translating a deeply embedded formula f' to a shallow embedding via $[\![\cdot]\!]$ (Definition 9) results in a trajectory formula that is satisfied by exactly the same set of sequences as f'. The proof is done by induction over the structure of trajectory formulas in the deep embedding.

Theorem 3. *Translate trajectory logic over Booleans in logic from deep to shallow embedding*

$$\forall f',\ \mathring{\sigma}\,.\ ([\![f']\!]\ \mathring{\sigma}\,) \Longleftrightarrow (\,\mathring{\sigma}\ \underline{\mathtt{sat}}\ f')$$

5 Wrapping It All Up

In this section we gather together the various theorems proved in Sections 3 and 4 to produce our final result. We then demonstrate the use of this result on the simple NAND-DELAY circuit first introduced in Section 1.

5.1 Gluing the Pieces Together

The focus of this paper is on the connection from trajectory evaluation over lattice values to relations over Boolean streams. Formalizing this connection forces us to reason about three different worlds: trajectory assertions with both lattice and Boolean values, and higher-order logic. A completely formal representation of these worlds (in particular, the semantics of higher-order logic [7]) and mappings between them would obfuscate the focus of our work. To maintain focus, we gloss over some of the semantic mappings between these different worlds. In particular, we use the same representation of trajectory formulas for both the metalogical results and a deep embedding of trajectory formulas in logic.

We link our shallow embedding of trajectory formulas in Section 4 with the results from Section 3 via an intermediate representation of trajectory logic that is deeply embedded in a higher-order logic. Theorem 4 says that a trajectory evaluation result over Boolean streams holds if and only if it holds in logic using a deep embedding of trajectory formulas.

Theorem 4. *Translate trajectory logic over Booleans to deep embedding in higher-order logic*

For all circuits ckt, antecedents ant, and consequences cons, if

For all ϕ.
  $\phi \models e$ **implies**
    For all $\mathring{\sigma}\,$.
      $\phi \models \left(\mathring{\sigma} \in \mathring{\mathcal{L}}\ ckt\right)$ **implies**
        For all $t \geq 0.\ \left((\phi,\, \mathring{\sigma}_t) \mathrel{\mathring{\models}} ant\right)$ **implies** $\left((\phi,\, \mathring{\sigma}_t) \mathrel{\mathring{\models}} cons\right)$

then the following is a true formula in HOL:

$$\models_{\mathsf{HOL}} \left(e \Longrightarrow \forall \mathring{\sigma}\,.\ \mathring{\sigma}\ \underline{\mathtt{in\_lang}}\ ckt \Longrightarrow \forall t \geq 0.\ (\mathring{\sigma}_t\ \underline{\mathtt{sat}}\ ant) \Longrightarrow (\mathring{\sigma}_t\ \underline{\mathtt{sat}}\ cons)\right)$$

We now have the pieces in place to prove a relationship between the standard trajectory logic and our shallow embedding of trajectory logic as predicates over Boolean streams in HOL (Theorem 5). This is proved by connecting Theorems 1, 2, 3, and 4.

Theorem 5. *Translate STE result to shallow embedding of Boolean streams. For all circuits ckt, antecedents ant, and consequences cons, if an implementation of the STE algorithm returns e:*

$$e = \mathtt{STE}\ ckt\ ant\ cons$$

then we can introduce the following axiom in HOL:

$$\vdash \left(e \Longrightarrow \forall \mathring{\sigma}.\ ckt\ \ \mathring{\sigma}\ \Longrightarrow \forall t \geq 0.\ [\![ant]\!]\ \ (\mathring{\sigma}_t) \Longrightarrow [\![cons]\!]\ \ (\mathring{\sigma}_t)\right)$$

The proof of Theorem 5 requires going through some simple transitivity reasoning to work from the STE algorithm to trajectory assertions (Theorem 1), unfolding $\models_{ckt} [ant \Longrightarrow\!\!> cons]$ (Definition 5) and using Theorem 2 to arrive at trajectory assertions over Boolean streams, and then to a deep embedding of trajectory logic in HOL (Theorem 4). We then use Theorem 3 to unify the right-hand-sides of Theorem 4 and Theorem 5. The unification is done by instantiating Theorem 3 first with "f'" as *ant* and then with "f'" as *cons*. Finally, we use the axioms relating the various representations of the circuit to prove that the sequence $\mathring{\sigma}$ is in the language of the circuit.

5.2 Simple Example Revisited

We now revisit the simple NAND-DELAY circuit first introduced in Section 1. When we introduced the circuit, we showed an intuitive, but not quite technically correct, translation from a standard trajectory assertion to a relations over Boolean streams result. We now demonstrate how the semantic link provided by our principal theorem (Theorem 5) induces a formally correct and intuitively satisfying connection between user-level verifications in trajectory evaluation and higher-order logic.

We begin with the trajectory formula that first appeared in Figure 2.

$$i \overset{def}{=} \texttt{"}i\texttt{"}$$
$$o \overset{def}{=} \texttt{"}o\texttt{"}$$
$$\text{NAND-DELAY} \models \left[\begin{matrix}(i \ \mathsf{is}\ b_1)\ \mathsf{and} \\ (o\ \mathsf{is}\ b_2)\end{matrix} \quad \Longrightarrow\!\!> (\mathsf{N}\ (o\ \mathsf{is}\ \neg(b_1 \wedge b_2)))\right]$$

Running the STE algorithm establishes that the NAND-DELAY circuit does satisfy the specification, and indeed, satisfies it for all valuations of the Boolean variables b_1 and b_2. We instantiate e with $\mathtt{T}$ in Theorem 5, and, after some unfolding and beta-reduction, we are left with the following.

$$
\begin{array}{l}
\vdash \forall \mathring{\sigma}\,. \\
\quad \text{NAND-DELAY } \mathring{\sigma} \implies \\
\quad \forall t \geq 0. \\
\qquad \begin{pmatrix} (\mathring{\sigma} \text{ ``}i\text{'' } t) = b_1 \wedge \\ (\mathring{\sigma} \text{ ``}o\text{'' } t) = b_2 \end{pmatrix} \implies \\
\qquad ((\mathring{\sigma} \text{ ``}o\text{'' } (t+1)) = \neg(b_1 \wedge b_2))
\end{array}
$$

Because b_1 and b_2 are free variables, we can universally quantify over them, and then use the equalities in the antecedent to substitute for b_1 and b_2 through the consequent. This finally leaves us with the result we had hoped for, namely the following intuitive user-level HOL theorem.

$$
\begin{array}{l}
\vdash \forall \mathring{\sigma}\,. \\
\quad \text{NAND-DELAY } \mathring{\sigma} \implies \\
\quad \forall t \geq 0. \\
\qquad ((\mathring{\sigma} \text{ ``}o\text{'' } (t+1)) = \neg\,((\mathring{\sigma} \text{ ``}i\text{'' } t) \wedge (\mathring{\sigma} \text{ ``}o\text{'' } t)))
\end{array}
$$

6 Voss Implementation of Trajectory Evaluation

In this section we briefly touch on the Voss implementation of trajectory logic. Information on Voss has been published previously [8], we include this section to make the connection to an implementation of trajectory evaluation more concrete. In Voss, the four-valued lattice is represented by a pair of Booleans, called a dual-rail value (Definition 10). In Voss, Booleans are implemented as BDDs and may therefore be symbolic. A Boolean value v is translated to a dual-rail value by putting v on the high rail and its negation on the low rail ($v{\downarrow} \overset{def}{=} (v, \neg v)$).

Definition 10. *Implementation of dual-rail lattice in Voss*

$$
\begin{array}{rcl}
\top & \overset{def}{=} & (\mathtt{F}, \mathtt{F}) \\
\mathsf{1} & \overset{def}{=} & (\mathtt{T}, \mathtt{F}) \\
\mathsf{0} & \overset{def}{=} & (\mathtt{F}, \mathtt{T}) \\
\mathsf{X} & \overset{def}{=} & (\mathtt{T}, \mathtt{T})
\end{array}
$$

Trajectory formulas are represented in Voss by a very simple, but slightly indirect, deep embedding. Rather than representing trajectory formulas by a data type that mirrors the abstract syntax tree of the formulas, a trajectory formula is a list of "5-tuples" (Figure 5). The five elements of each tuple are the guard, a node name, a Boolean value (which may be a variable), a natural-number start-time, and a natural-number end-time. The meaning is "if the guard is true, then the node takes on the given value from the start-time to the end-time."

As the definitions in Figure 5 show, every formula in the trajectory logic can be represented quite simply by a 5-tuple list. Moreover, it is clear that the list representation does not add anything new to the expressive power of our formulas: any 5-tuple whose duration spans more than one-time unit can be expressed by an appropriate conjunction of applications of the next time operator.

$$
\begin{array}{llllllll}
// & & \textit{guard} & \textit{node} & \textit{value} & \textit{start} & \textit{end} & \\
\texttt{traj\_form} & \overset{\textit{type}}{=} & (\ \texttt{bool}\ \times & \texttt{string} \times & \texttt{bool}\ \times & \texttt{nat}\ \times & \texttt{nat}\ ) & \texttt{list}
\end{array}
$$

$$
\begin{array}{lll}
n \ \mathsf{is}\ v & \overset{\textit{def}}{=} & [(\mathtt{T}, n, v, 0, 1)] \\
f_1 \ \mathsf{and}\ f_2 & \overset{\textit{def}}{=} & f_1 \ \texttt{append}\ f_2 \\
f \ \ \mathsf{when}\ g & \overset{\textit{def}}{=} & \texttt{map}\ (\lambda(g', n, v, t_0, t_1).\ (g' \wedge g, n, v, t_0, t_1))\ f \\
\mathsf{N}\ f & \overset{\textit{def}}{=} & \texttt{map}\ (\lambda(g, n, v, t_0, t_1).\ (g, n, v, t_0 + 1, t_1 + 1))\ f \\
 & & \\
f \ \mathsf{from}\ t_0 & \overset{\textit{def}}{=} & \texttt{map}\ (\lambda(g, n, v, z, t_1).\ (g, n, v, t_0, t_1))\ f \\
f \ \mathsf{to}\ t_1 & \overset{\textit{def}}{=} & \texttt{map}\ (\lambda(g, n, v, t_0, z).\ (g, n, v, t_0, t_1))\ f
\end{array}
$$

Fig. 5. Implementation of trajectory formulas in Voss

7 Conclusion

The motivation of our work is to combine the automatic deductive power of finite-state model checking with the expressive power and flexibility of general-purpose theorem proving. Key elements of such an architecture are semantic links between the general logic of the theorem prover and the more specialised model-checker logic. To this end, we have established a semantic link between correctness results in the temporal logic of symbolic trajectory evaluation and relational hardware specifications in higher order logic.

Ultimately, our aim is to combine results from several different model checkers in a single system. The expressive power of typed higher-order logic, which is a foundational formalism, and the generality of the relational approach are an excellent "glue logic" for this. For many other linear-time temporal logics, at least, it seems quite reasonable to expect smooth embeddings into the relational world. Indeed, there have been several such embeddings in the literature [18,4]. Branching time logics may be more challenging, but the PVS results cited earlier are encouraging. More general temporal logics, such as Extended Temporal Logic [17] will provide a further challenge.

Acknowledgments

This paper benefited from the helpful comments of Carl Seger, Ching-Tsun Chou, and the anonymous referees.

References

1. M. D. Aagaard, R. B. Jones, and C.-J. H. Seger. Formal verification using parametric representations of Boolean constraints. In *ACM/IEEE Design Automation Conference*, July 1999. 202
2. S. Agerholm and H. Skjødt. Automating a model checker for recursive modal assertions in HOL. Technical Report DAIMI IR-92, Computer Science Department, Aarhus University, 1990. 205
3. A. Cheng and K. Larsen, editors. *Program and Abstracts of the BRICS Autumn School on the Verification*, Aug. 1996. BRICS Notes Series NS-96-2. 202
4. C.-T. Chou. Predicates, temporal logic, and simulations. In C.-J. H. Seger and J. J. Joyce, editors, *HOL User's Group Workshop*, pages 310–323. Springer Verlag; New York, Aug. 1994. 217
5. C.-T. Chou. The mathematical foundation of symbolic trajectory evaluation. In *Workshop on Computer-Aided Verification*. Springer Verlag; New York, 1999. *To appear.* 206
6. G. C. Gopalakrishnan and P. J. Windley, editors. *Formal Methods in Computer-Aided Design*. Springer Verlag; New York, Nov. 1998. 202
7. M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: a theorem proving environment for higher order logic*. Cambridge University Press, New York, 1993. 203, 205, 214
8. S. Hazelhurst and C.-J. H. Seger. Symbolic trajectory evaluation. In T. Kropf, editor, *Formal Hardware Verification*, chapter 1, pages 3–78. Springer Verlag; New York, 1997. 205, 206, 216
9. A. J. Hu and M. Y. Vardi, editors. *Computer Aided Verification*. Springer Verlag; New York, July 1998. 202
10. J. Joyce and C.-J. Seger. Linking BDD based symbolic evaluation to interactive theorem proving. In *ACM/IEEE Design Automation Conference*, June 1993. 204, 205
11. M. Newey and J. Grundy, editors. *Theorem Proving in Higher Order Logics*. Springer Verlag; New York, Sept. 1998. 202
12. J. O'Leary, X. Zhao, R. Gerth, and C.-J. H. Seger. Formally verifying IEEE compliance of floating-point hardware. *Intel Technical Journal*, First Quarter 1999. Online at http://developer.intel.com/technology/itj/. 202
13. The omega project, 1999. http://www.ags.uni-sb.de/projects/deduktion/.
14. Proof and specification assisted design environments, ESPRIT LTR project 26241, 1999. *http://www.dcs.gla.ac.uk/prosper/*.
15. S. Rajan, N. Shankar, and M. Srivas. An integration of model checking automated proof checking. In *Workshop on Computer-Aided Verification*. Springer Verlag; New York, 1996. 205
16. C.-J. H. Seger and R. E. Bryant. Formal verification by symbolic evaluation of partially-ordered trajectories. *Formal Methods in System Design*, 6(2):147–189, Mar. 1995. 202, 209
17. M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115:1–37, 1994. 217
18. J. von Wright. Mechanising the temporal logic of actions in HOL. In M. Archer, J. J. Joyce, K. N. Levit, and P. J. Windley, editors, *International Workshop on the HOL Theorem Proving System and its Applications*, pages 155–159. IEEE Computer Society Press, Washington D. C., Aug. 1991. 217