A Compositional Approach to CTL* Verification*

Yonit Kesten[1] and Amir Pnueli[2]

[1] Ben Gurion University, ykesten@bgumail.bgu.ac.il* **
[2] Weizmann Institute of Science, amir@wisdom.weizmann.ac.il

Abstract. The paper presents a compositional approach to the verification of CTL* properties over reactive systems. Both symbolic model-checking (SMC) and deductive verification are considered. Both methods are based on two decomposition principles. A *general state formula* is decomposed into *basic state formulas* which are CTL* formulas with no embedded path quantifiers. To deal with *arbitrary* basic state formulas, we introduce another reduction principle which replaces each *basic path formula*, i.e., path formulas whose principal operator is temporal and which contain no embedded temporal operators or path quantifiers, by a newly introduced boolean variable which is added to the system. Thus, both the algorithmic and the deductive methods are based on two *statification* transformations which successively replace temporal formulas by assertions which contain no path quantifiers or temporal operators. Performing these decompositions repeatedly, we remain with *basic assertional formulas*, i.e., formulas of the form $E_f p$ and $A_f p$ for some assertion p. In the model-checking method we present a single symbolic algorithm to verify both universal and existential basic assertional properties. In the deductive method we present a small set of proof rules and show that this set is sound and relatively complete for verifying universal and existential basic assertional properties over reactive systems. Together with two proof rules for the decompositions, we obtain a sound and relatively complete proof system for arbitrary CTL* properties. Interestingly, the deductive approach for CTL* presented here, offers a viable new approach to the deductive verification of arbitrary LTL formulas.

1 Introduction

The paper presents a compositional approach to the verification of CTL* properties over reactive systems. We present both an algorithmic method based on symbolic model-checking, for the verification of finite-state systems and a deductive method for the verification of possibly infinite-state systems.

The logic CTL* is a temporal logic which can express linear-time as well as branching-time temporal properties, and combinations thereof, and contains

*** Contact author

* This research was supported in part by the Minerva Center for Verification of Reactive Systems, the Israel Science Foundation (grant no. 106/02-1), and NSF grant CCR-0205571.

both LTL and CTL as sub-logics. A complete deductive proof system for linear-time temporal logic (LTL) has been presented in [15] and further elaborated in [16] and [17]. This proof system has been successfully implemented in the Stanford Temporal Verifier STeP [1]. The deductive proof system presented here can be viewed as an extension of the approach of [15] to the logic CTL*.

A deductive proof system for CTL* is valuable for several reasons. In spite of the impressive progress in the various versions of model-checking and other algorithmic verification techniques, they are still restricted to finite-state systems. The only verification method known to be complete for all programs is still the method of deductive verification. There are special benefits to the extension of the deductive methodology from the linear-time framework to the more expressive branching semantics of CTL*:

1. Some important system properties are expressible in CTL* but not in LTL. Typically, these are "possibility" properties, such as the *viability* of a system, stating that any reachable state can spawn a fair computation. This is strongly related to the non-zeno'ness of real-time and hybrid systems.
2. As shown in [21], the problem of synthesis of a reactive module can be solved by checking for validity of a certain CTL* formula, even if the original specification is a pure LTL formula.

Deductive verification of CTL* formulas is valuable even in the context of finite-state systems which can be model-checked:

3. A counter-example of even simple CTL formulas such as $E\,\Box\, p$ is no longer a simple finite printable trace. A convincing evidence of a counter-example could be an automatically produced *proof* of its existence.
4. In general, model-checking is useful if it produces a counter-example. However, when it terminates declaring the property to be valid, the user is not always convinced. A *deductive proof* can provide a convincing argument of such a validity [18, 19].

A general CTL* formula is composed of state sub-formulas that are interpreted over states and path sub-formulas that are interpreted over paths. Both the algorithmic and the deductive methods for CTL* verification are based on the same basic decomposition principles. A *general state formula* is decomposed into *basic state formulas* which are CTL* formulas with no embedded path quantifiers. A basic state formula has the form $\mathcal{Q}\varphi$, where $\mathcal{Q}$ is a path quantifier and φ is a *general path formula* (according to the CTL* terminology) or, equivalently, can be described as an LTL formula. A *general path formula* is decomposed into *basic path formulas* which are path formulas whose principal operator is temporal and which contain no embedded temporal operators or path quantifiers.

Thus, as a first step, we reduce the problem of verifying an arbitrary CTL* formula into a set of verification tasks of the form $p_i \Rightarrow \beta_i$ in the deductive case ($p_i \Leftrightarrow \beta_i$ in the algorithmic case), where p_i is an assertion and β_i is a basic state formula. This reduction is based on the following observation: Let $f(\beta)$ be a CTL* formula which contains one or more occurrences of the basic state formula

β. Then, a sufficient condition for $f(\beta)$ to be valid over the computation tree of system $\mathcal{D}$ ($\mathcal{D}$-valid) is the $\mathcal{D}$-validity of the formulas $p \Rightarrow \beta$ and $f(p)$, for some assertion p, where $f(p)$ is obtained from $f(\beta)$ by replacing all occurrences of β by the assertion p. By repeated application of such replacements (for appropriately designed assertions p), we reduce the verification problem of an arbitrary CTL* formula to a set of verification problems, each requiring the proof of a formula of the form $p_i \Rightarrow \beta_i$.

In the context of finite-state model checking, this decomposition of the verification task based on the path quantifiers has been first proposed by Emerson and Lei in [6]. It has been used again in [11] for the construction of a symbolic model checker (SMC) for CTL* properties over finite state systems.

To deal with *arbitrary* basic state formulas, we introduce another reduction principle which replaces each *basic path formula* by a newly introduced boolean variable which is added to the system $\mathcal{D}$. This reduction can be viewed as a simplified version of the tableau construction proposed in [14] and later referred to as the construction of a *tester* [10].

Thus, both the algorithmic and the deductive methods are based on two *statification* transformations which successively replace temporal formulas by assertions which contain no path quantifiers or temporal operators. The first transformation replaces a basic state formula β by an assertion p, provided that we can independently establish the $\mathcal{D}$-validity of the entailment $p \Rightarrow \beta$. The second transformation replaces the basic path formula φ by the single boolean variable x_φ (which is also a trivial assertion) at the price of augmenting the system $\mathcal{D}$ by a temporal tester T_φ.

It is interesting to compare the general structure of the CTL* deductive proof system presented here with the LTL proof system presented in [15] and elaborated in [16, 17, 1]. In [15], the system lists first useful rules for special form formulas, the most important of which are formulas of the form $p \Rightarrow \Box q$, $p \Rightarrow \Diamond q$, and $\Box \Diamond p \Rightarrow \Box \Diamond q$, where p and q are arbitrary *past formulas*. To deal with the general case, [15] invokes a general canonic-form theorem, according to which every (quantifier-free) LTL formula is equivalent to a conjunction of formulas of the form $\Box \Diamond p_i \Rightarrow \Box \Diamond q_i$, for some past formulas p_i and q_i. While this approach is theoretically adequate, it is not a practically acceptable solution to the verification of arbitrary LTL formulas. This is because the best known algorithms for converting an arbitrary LTL formula into canonic form are at least exponential (e.g., [7] which is actually non-elementary). A better approach to the verification of arbitrary LTL formulas is based on the notion of *deductive model checking* [23], which can also be described as a tableau-based construction.

The approach presented here, gives a small set of proof rules that is sound and relatively complete for proving basic assertional formulas, which are formulas of the form $\mathcal{Q}p$ where $\mathcal{Q}$ is a path quantifier and p is an assertion. To deal with the general case, a general path formula is decomposed to basic path formulas based on successive elimination of temporal operators. This decomposition, which proceeds all the way to basic assertional formulas, can be viewed as an incremental tableau construction which offers a viable new approach to the

deductive verification of arbitrary LTL formulas, even though it is presented as part of a deductive proof system for CTL*, which is a more complex logic than LTL.

There have been two earlier complete axiomatizations of propositional CTL*. The work reported in [22] provides (the first) complete axiomatization of pure propositional CTL*, thus solving a long standing open problem in branching-time temporal logic. Comparing this impressive result with our work, we should remind the reader of our motivation which is to provide a deductive system to verify first-order CTL* expressible properties of reactive systems, where the computational model includes a full fledged set of weak and strong fairness assumptions. Our goal is to derive a deductive system for CTL* which extends the LTL deductive methodology expounded in [17] and provides realistic tools for verifying non-trivial reactive systems, such as those implemented in the STeP system [1]. Theoretically, this goal can be implemented even within the pure-logic axiomatization of [22], because CTL* (being more expressive than LTL) can certainly capture the computation tree of a reactive system including all fairness assumptions. This allows us to reduce the verification problem $\mathcal{D} \models \varphi$ into the pure validity problem $\models S_{\mathcal{D}} \rightarrow \varphi$, where $S_{\mathcal{D}}$ is the CTL* formula characterizing the computation tree of system $\mathcal{D}$. While this is possible in theory, it is highly impractical and leads to very obscure and unreadable proofs. A similar dichotomy exists in finite-state LTL verification. On one hand, one can use the special LTL model checking technique proposed in [14] for verifying $\mathcal{D} \models \varphi$ whose complexity is exponential in φ but only linear in $\mathcal{D}$. On the other hand, one can reduce this to the problem of checking the validity of the implication $S_{\mathcal{D}} \rightarrow \varphi$ which is exponential in both φ and $\mathcal{D}$. It is obvious that the specialized technique which does not transform the system into a formula is (exponentially) better than the reductionist approach. While the analogy between finite-state model checking and deductive verification is not perfect, this argument serves to indicate the inherent rise in complexity when using pure temporal logic techniques for practical verification.

Another related work is that of Sprenger [24]. This approach is much closer to our own, because it preserves the distinction between the system and the formula, and contains a special treatment of the different kinds of fairness. The advantage of our approach is that it proceeds at a coarser level of granularity, and therefore yields a much simpler proof system. Sprenger's method of local model checking proceeds at steps analogous to the basic steps of a tableau construction, including step by step handling of the boolean connectives. Our approach attempts to get rid of one temporal operator at each step, applying the appropriate rule for this operator, with no need to trace cycles and close leaves in the tableau. We believe that our proof system and its application to be significantly more succinct and effective and, therefore, more amenable to the construction of support systems for serious reactive verification.

The paper is organized as follows. In Section 2 we present the FDS computation model. In Section 3, we present the logic CTL* with past operators. In Section 4 we present the notion of a *tester* which is an FDS constructed for a basic

path formula (i.e., a simple LTL formula) and identifies the positions in a computation at which the formula holds. Testers are constructs which are central to our method of decomposing a verification of a large CTL* formula into subtasks of verifying smaller sub-formulas. In Section 5 we present a method for model checking an arbitrary CTL* formula, using the decomposition principles and the single FEASIBLE algorithm used to verify basic assertional formulas. Finally, in Section 6 we present a deductive proof system for the verification of general CTL* formulas. The presentation starts by presenting a proof rule for the elimination of state formulas embedded within a general CTL* formula. Next, we present a set of proof rules for some basic state properties which is sound and relatively complete for verifying basic assertional formulas. Finally we give a proof rule for the elimination of temporal operators from an arbitrary state formula, using a tester for this purpose We prove soundness and relative completeness of the deductive system for verifying arbitrary CTL* formulas over reactive systems.

The algorithmic method for CTL* verification has been presented in [11] and is given here for the completeness of the presentation. A conference version of the deductive method appeared in [9].

It should be noted that both the algorithmic and deductive methods for CTL* verification are presented here in terms of a decomposition of an arbitrary CTL* formula all the way to basic assertional formulas. Obviously this is not the most efficient method, and is presented here mainly for the simplicity of the presentation. For better efficiency, the decomposition of a general state formula should stop at CTL formulas, namely refrain from eliminating the outermost temporal operator. In the algorithmic method, this requires augmenting the proof system with a set of algorithms for all CTL operators. This option is discussed in [20]. In the deductive method, the proof system has to be augmented to be sound and relatively complete for CTL.

2 The Computational Model

As a computational model for reactive systems, we take the model of *fair discrete system* (FDS). An FDS $\mathcal{D}: \langle V, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$ consists of the following components.

- $V = \{u_1, ..., u_n\}$: A finite set of typed *state variables* over possibly infinite domains. We define a *state* s to be a type-consistent interpretation of V, assigning to each variable $u \in V$ a value $s[u]$ in its domain. We denote by Σ the set of all states.
- Θ : The *initial condition*. This is an assertion characterizing all the initial states of the FDS. A state is called *initial* if it satisfies Θ.
- ρ : A *transition relation*. This is an assertion $\rho(V, V')$, relating a state $s \in \Sigma$ to its $\mathcal{D}$-successor $s' \in \Sigma$ by referring to both unprimed and primed versions of the state variables. The transition relation $\rho(V, V')$ identifies state s' as a *$\mathcal{D}$-successor* of state s if $\langle s, s' \rangle \models \rho(V, V')$, where $\langle s, s' \rangle$ is the joint interpretation which interprets $x \in V$ as $s[x]$, and x' as $s'[x]$.

- $\mathcal{J} = \{J_1, \ldots, J_k\}$: A set of assertions expressing the *justice* (*weak fairness*) requirements. Intentionally, the justice requirement $J \in \mathcal{J}$ stipulates that every computation contains infinitely many J-states (states satisfying J).
- $\mathcal{C} = \{\langle p_1, q_1\rangle, \ldots \langle p_n, q_n\rangle\}$: A set of assertions expressing the *compassion* (*strong fairness*) requirements . Intentionally, the compassion requirement $\langle p, q\rangle \in \mathcal{C}$ stipulates that every computation containing infinitely many p-states also contains infinitely many q-states.

Let $\sigma : s_0, s_1, ...,$ be a sequence of states, φ be an assertion, and $j \geq 0$ be a natural number. We say that j is a φ-position of σ if s_j is a φ-state, i.e., s_j satisfies the assertion φ. For the case that σ is infinite, we define $|\sigma| = \omega$ and the range $[m..|\sigma|)$ denotes the set of all integers $j \geq m$. For the case that $\sigma : s_0, \ldots, s_k$, $|\sigma| = k + 1$ and the range $[m..|\sigma|)$ denotes the set of all integers j, $m \leq j \leq k$.

Let $\mathcal{D}$ be an FDS for which the above components have been identified. A *run* of $\mathcal{D}$ is a *maximal* sequence of states $\sigma : s_0, s_1, ...,$ satisfying the requirements of

- *Initiality:* s_0 is initial, i.e., $s_0 \models \Theta$.
- *Consecution:* For each $j + 1 \in [1..|\sigma|)$, s_{j+1} is a $\mathcal{D}$-successor of s_j.

The sequence σ being maximal means that either σ is infinite, or $\sigma = s_0, \ldots, s_k$ and s_k has no $\mathcal{D}$-successor.

We denote by $runs(\mathcal{D})$ the set of runs of $\mathcal{D}$. An infinite run of $\mathcal{D}$ is called a *computation* if it satisfies the following:

- *Justice:* For each $J \in \mathcal{J}$, σ contains infinitely many J-positions
- *Compassion:* For each $\langle p, q\rangle \in \mathcal{C}$, if σ contains infinitely many p-positions, it must also contain infinitely many q-positions.

We denote by $Comp(\mathcal{D})$ the set of all computations of $\mathcal{D}$.

A state s is said to be *reachable* if it participates in some run of $\mathcal{D}$. State s is *feasible* if it participates in some computation of $\mathcal{D}$. An FDS $\mathcal{D}$ is called *deadlock-free* if every reachable state has a $\mathcal{D}$-successor. Note that all runs of a deadlock-free FDS are infinite. It can be shown that all FDS's derived from programs are deadlock-free. An FDS $\mathcal{D}$ is *feasible* if it has at least one computation, i.e., if $Comp(\mathcal{D}) \neq \emptyset$. We say that an FDS $\mathcal{D}$ is *viable* if every reachable state is feasible. It is not difficult to see that every viable FDS is deadlock-free.

Parallel Composition of FDS's

Fair discrete systems can be composed in parallel. Let $\mathcal{D}_i = \langle V_i, \Theta_i, \rho_i, \mathcal{J}_i, \mathcal{C}_i\rangle$, $i \in \{1, 2\}$, be two fair discrete systems. Two versions of parallel composition are used. Asynchronous composition is used to assemble an asynchronous system from its components (see [8]). We define the *asynchronous* parallel composition of two FDS's to be

$$\mathcal{D} = \langle V, \Theta, \rho, \mathcal{J}, \mathcal{C}\rangle \quad = \quad \langle V_1, \Theta_1, \rho_1, \mathcal{J}_1, \mathcal{C}_1\rangle \parallel \langle V_2, \Theta_2\rho_2, \mathcal{J}_2, \mathcal{C}_2\rangle,$$

where

$$V = V_1 \cup V_2 \qquad \Theta = \Theta_1 \wedge \Theta_2,$$
$$\rho = \rho_1 \ \wedge \ pres(V_2 - V_1) \ \vee \ \rho_2 \ \wedge \ pres(V_1 - V_2),$$
$$\mathcal{J} = \mathcal{J}_1 \cup \mathcal{J}_2, \quad \mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2.$$

For a set of variables U, the predicate $pres(U)$ is an abbreviation of the conjunction $\bigwedge_{x \in U} (x' = x)$, implying that all variables in U preserve their values in the current transition.

The execution of $\mathcal{D} = \mathcal{D}_1 \| \mathcal{D}_2$ is the *interleaved execution* of $\mathcal{D}_1$ and $\mathcal{D}_2$.

Another mode of parallel composition is that of *synchronous parallel composition*. Synchronous composition is used in some cases, to assemble a system from its components (in particular when considering hardware designs which are naturally synchronous). However, our primary use of synchronous composition is for combining a system with a *tester* T_φ for a basic path formula φ, as described, for example, in Subsection 5.4. We define the *synchronous* parallel composition of two FDS's to be

$$\mathcal{D} = \langle V, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle \quad = \quad \langle V_1, \Theta_1, \rho_1, \mathcal{J}_1, \mathcal{C}_1 \rangle \ ||| \ \langle V_2, \Theta_2 \rho_2, \mathcal{J}_2, \mathcal{C}_2 \rangle,$$

where

$$V = V_1 \cup V_2 \qquad \Theta = \Theta_1 \wedge \Theta_2, \quad \rho = \rho_1 \wedge \rho_2,$$
$$\mathcal{J} = \mathcal{J}_1 \cup \mathcal{J}_2, \quad \mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2.$$

We can view the execution of $\mathcal{D}$ as the *joint execution* of $\mathcal{D}_1$ and $\mathcal{D}_2$.

From FDS to JDS

An FDS with no compassion requirements is called a *just discrete system* (JDS).

Let $\mathcal{D} : \langle V, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$ be an FDS such that $\mathcal{C} = \{(p_1, q_1), \ldots, (p_m, q_m)\}$ and $m > 0$. We define a JDS $\mathcal{D}_J : \langle V_J, \Theta_J, \rho_J, \mathcal{J}_J, \emptyset \rangle$ equivalent to $\mathcal{D}$, as follows:

- $V_J = V \cup \{n\_p_i : \textbf{boolean} \mid (p_i, q_i) \in \mathcal{C}\} \cup \{x_c\}$. That is, for every compassion requirement $(p_i, q_i) \in \mathcal{C}$, we add to V_J a boolean variable n_p_i. Variable n_p_i is intended to turn true at a point in a computation from which the assertion p_i remains false forever. Variable x_c, common to all compassion requirements, is intended to turn true at a point in a computation satisfying $\bigvee_{i=1}^{m} (p_i \wedge n\_p_i)$.
- $\Theta_J = \Theta \ \wedge \ x_c = 0 \ \wedge \ \bigwedge_{(p_i, q_i) \in \mathcal{C}} n\_p_i = 0.$
  That is, initially all the newly introduced boolean variables are set to zero.
- $\rho_J = \rho \ \wedge \ \rho_{n\_p} \ \wedge \ \rho_c$, where

$$\rho_{n\_p} : \bigwedge_{(p_i, q_i) \in \mathcal{C}} (n\_p_i \rightarrow n\_p_i')$$

$$\rho_c \quad : x_c' = \left(x_c \ \vee \ \bigvee_{(p_i, q_i) \in \mathcal{C}} (p_i \ \wedge \ n\_p_i) \right)$$

The augmented transition relation allows each of the n_p_i variables to change non-deterministically from 0 to 1. Variable x_c is set to 1 on the first occurrence of $p_i \wedge n\_p_i$, for some i, $1 \geq i \geq m$. Once set, it is never reset.

- $\mathcal{J}_J = \mathcal{J} \cup \{\neg x_c\} \cup \{n\_p_i \vee q_i \mid (p_i, q_i) \in \mathcal{C}\}$.
  The augmented justice set contains the additional justice requirement $n\_p_i \vee q_i$ for each $(p_i, q_i) \in \mathcal{C}$. This requirement demands that either n_p_i turns true sometime, implying that p_i is continuously false from that time on, or that q_i holds infinitely often.
  The justice requirement $\neg x_c$ ensures that a run with one of the variables n_p_i set prematurely, will not be accepted as a computation.

The transformation of an FDS to a JDS follows the transformation of Streett automata to generalized Büchi Automata (see [2] for finite state automata, [26] for infinite state automata). The reactive systems we want to verify can have both weak (justice) and strong (compassion) fairness requirements. In the case of algorithmic verification we use an algorithm that deals with the compassion requirements of the system directly. We show in [11] that this approach is more efficient. However in the deductive case, although [15] presents an LTL proof rule that deals directly with compassion, and can easily be adapted for CTL*, the application of the rule initiates a tree of additional rules to be included, resulting in a complex proof system. To keep the presentation reasonable, we prefer in the exposition of the deductive approach to transform the FDS representation of the verified system into a JDS, prior to the verification, and present rules which only deal with justice.

3 The Branching Temporal Logic CTL*

In the following we define the branching temporal logic CTL* with both future and past operators. We denote the fragment of CTL* without the past operators as the *future fragment* of CTL* (see [4] for the future fragment). We assume a finite set of variables V over possibly infinite domains, and an underlying assertion language $\mathcal{L}$ which contains the predicate calculus augmented with fix-point operators[3]. We assume that $\mathcal{L}$ contains interpreted symbols for expressing the standard operations and relations over some concrete domains, such as the integers.

In practice, our assertional language is a first-order language over the integers. An *assertion* is a formula in that language. We will rarely use fix-point operators in assertions and their inclusion is merely for the sake of claims of relative completeness.

A CTL* formula is constructed out of assertions to which we apply the boolean operators, temporal operators and path quantifiers. The basic temporal operators are

[3] As is well known ([13],) a first-order language is not adequate for (relative) completeness of a temporal proof system for infinite state reactive programs. The use of minimal and maximal fix-points for relative completeness of the liveness rules is discussed in [15], based on [25].

$\bigcirc$ – Next $\quad \ominus$ – Previous $\quad \circledcirc$ – Weak Previous
$\mathcal{U}$ – Until $\quad \mathcal{S}$ – Since
$\mathcal{W}$ – Waiting-for (Unless) $\quad \mathcal{B}$ – Back-to
Additional temporal operators may be defined as follows:

$\Diamond p = \mathrm{T}\mathcal{U}p$ – Eventually p
$\Box p = p\mathcal{W}\mathrm{F}$ – Always, henceforth p
$\diamondsuit p = \mathrm{T}\mathcal{S}p$ – Sometimes in the past p
$\boxminus p = p\mathcal{B}\mathrm{F}$ – Always in the past p

The path quantifiers are E, A, E_f and A_f. We refer to E_f and A_f as the *fair* path quantifiers and to E and A as the *unrestricted* path quantifiers. In the following, we present the syntax and semantics of the logic which is interpreted over the computation graph generated by an FDS. We use the terms *path* and *fair path* as synonymous to a *run* and a *computation* respectively, over an FDS. Let $\pi : s_0, s_1, \ldots$ be a run of $\mathcal{D}$. Then, we write $\pi[j]$ to denote s_j, the j'th state in π.

The Logic CTL*

There are two types of sub-formulas in CTL*: *state formulas* that are interpreted over states, and *path formulas* that are interpreted over paths. The syntax of a CTL* formula is defined inductively as follows.
State formulas:

- Every assertion in $\mathcal{L}$ is a state formula.
- If p is a *path formula*, then Ep, Ap, E_fp and A_fp are state formulas.
- If p and q are state formulas then so are $p \vee q$ and $p \wedge q$.

Path formulas:

- Every state formula is a path formula.
- If p and q are path formulas then so are $p \vee q$, $p \wedge q$, $\bigcirc p$, $p\mathcal{U}q$, $p\mathcal{W}q$, $\ominus p$, $\circledcirc p$, $p\mathcal{S}q$, and $p\mathcal{B}q$.

The formulas of CTL* are all the state formulas generated by the above rules.

A formula of the form $\mathcal{Q}\psi$, where $\mathcal{Q} \in \{E, A, E_f, A_f\}$ is a path quantifier and ψ is a path formula containing no other path quantifier, is called a *basic state formula*. A basic state formula of the form $A\psi$ or $A_f\psi$ ($E\psi$ or $E_f\psi$) is called a basic *universal* (*existential*) state formula. A basic state formula $\mathcal{Q}p$ where p is an assertion is called a *basic assertional formula*.

We define a *basic path formula* to be a path formula φ whose principal operator is temporal, and such that φ contains no other temporal operators or path quantifiers. We refer to the set of variables that occur in a formula p as the *vocabulary of* p.

The semantics of a CTL* formula p is defined with respect to an FDS $\mathcal{D}$ over the vocabulary of p. The semantics is defined inductively as follows. State formulas are interpreted over states in $\mathcal{D}$. We define the notion of a CTL* formula p holding at a state s in $\mathcal{D}$, denoted $(\mathcal{D}, s) \models p$, as follows:

- For an assertion p,

$(\mathcal{D}, s) \models p \quad \Leftrightarrow \quad s \models p$

- For state formulas p and q,

$(\mathcal{D}, s) \models p \vee q \quad \Leftrightarrow \quad (\mathcal{D}, s) \models p$ or $(\mathcal{D}, s) \models q$

$(\mathcal{D}, s) \models p \wedge q \quad \Leftrightarrow \quad (\mathcal{D}, s) \models p$ and $(\mathcal{D}, s) \models q$

- For a path formula p,

$(\mathcal{D}, s) \models Ep \quad \Leftrightarrow \quad (\mathcal{D}, \pi, j) \models p$ for some path $\pi \in \mathit{runs}(\mathcal{D})$ and position $j \in [0..|\pi|)$ satisfying $\pi[j] = s$.

$(\mathcal{D}, s) \models Ap \quad \Leftrightarrow \quad (\mathcal{D}, \pi, j) \models p$ for all paths $\pi \in \mathit{runs}(\mathcal{D})$ and positions $j \in [0..|\pi|)$ satisfying $\pi[j] = s$.

The semantics of $E_f p$ and $A_f p$ are defined similarly to Ep and Ap respectively, replacing *path* (run) by *fair path* (fair runs or computations).

Path formulas are interpreted over runs of $\mathcal{D}$. We denote the notion of a CTL* formula p holding at position j of a run $\pi \in \mathit{runs}(\mathcal{D})$, by $(\mathcal{D}, \pi, j) \models p$. When $j = 0$, we use the shorthand notation $(\mathcal{D}, \pi) \models p$. The semantics of path formulas is defined as follows:

- For a state formula p,

$(\mathcal{D}, \pi, j) \models p \quad \Leftrightarrow \quad (\mathcal{D}, \pi[j]) \models p.$

- For path formulas p and q,

$(\mathcal{D}, \pi, j) \models p \vee q \quad \Leftrightarrow \quad (\mathcal{D}, \pi, j) \models p$ or $(\mathcal{D}, \pi, j) \models q$

$(\mathcal{D}, \pi, j) \models p \wedge q \quad \Leftrightarrow \quad (\mathcal{D}, \pi, j) \models p$ and $(\mathcal{D}, \pi, j) \models q$

$(\mathcal{D}, \pi, j) \models \bigcirc p \quad \Leftrightarrow \quad j + 1 < |\pi|$ and $(\mathcal{D}, \pi, j + 1) \models p$

$(\mathcal{D}, \pi, j) \models p \mathcal{U} q \quad \Leftrightarrow \quad (\mathcal{D}, \pi, k) \models q$ for some $k \in [j..|\pi|)$, and $(\mathcal{D}, \pi, i) \models p$ for all $i \in [j..k)$

$(\mathcal{D}, \pi, j) \models p \mathcal{W} q \quad \Leftrightarrow \quad (\mathcal{D}, \pi, j) \models p \mathcal{U} q$, or $(\mathcal{D}, \pi, i) \models p$ for all $i \in [j..|\pi|)$

$(\mathcal{D}, \pi, j) \models \ominus p \quad \Leftrightarrow \quad j > 0$ and $(\mathcal{D}, \pi, j-1) \models p$

$(\mathcal{D}, \pi, j) \models \circleddash p \quad \Leftrightarrow \quad j = 0$ or $(\mathcal{D}, \pi, j-1) \models p$

$(\mathcal{D}, \pi, j) \models p \mathcal{S} q \quad \Leftrightarrow \quad (\mathcal{D}, \pi, k) \models q$ for some k, $0 \leq k \leq j$ and $(\mathcal{D}, \pi, i) \models p$ for every i, $k < i \leq j$

$(\mathcal{D}, \pi, j) \models p \mathcal{B} q \quad \Leftrightarrow \quad (\mathcal{D}, \pi, j) \models p \mathcal{S} q$, or $(\mathcal{D}, \pi, i) \models p$ for every i, $0 \leq i \leq j$

Let p be a CTL* formula. We say that p *holds* on $\mathcal{D}$ (p is $\mathcal{D}$-valid), denoted $\mathcal{D} \models p$, if $(\mathcal{D}, s) \models p$, for every initial state s in $\mathcal{D}$. A CTL* formula p is called *satisfiable* if it holds on some model. A CTL* formula is called *valid* if it holds on all models.

We refer to a state which satisfies p as a *p-state*. Let p and q be CTL* formulas. We introduce the abbreviation

$$p \Rightarrow q \quad \text{for} \quad A \Box (p \rightarrow q).$$

where $p \rightarrow q$ is the logical implication equivalent to $\neg p \vee q$. Thus, the formula $p \Rightarrow q$ holds at $\mathcal{D}$ if the implication $p \rightarrow q$ holds at all reachable states.

Let V be a set of variables and ψ be a CTL* formula over V. We denote by ψ' the formula ψ in which every variable $v \in V$ is replaced by the primed variable v'.

The restricted subset of CTL* in which each temporal operator is immediately preceded by a path quantifier is called *Computation Tree Logic* (CTL). A

state formula whose principal operators are a pair $\mathcal{QT}$ (where $\mathcal{Q}$ is a path quantifier and $\mathcal{T}$ is a temporal operator) and which does not contain any additional temporal operators or path quantifiers is called a *basic* CTL *formula*.

We refer to path formulas which do not contain any path quantifiers as LTL formulas.

The Verification Problem

Having presented our model for systems and the specification language of CTL*, we can formulate the problem this paper intends to solve.
Given a viable FDS $\mathcal{D}$ and a CTL* formula φ, our goal is to verify that φ is $\mathcal{D}$-valid, i.e., all computations of $\mathcal{D}$ satisfy φ.

With no loss of generality, we assume that formula φ is given in *positive normal form*, i.e. contains negations only as part of assertions. It is straightforward to transform an arbitrary CTL* formula ψ into a positive form CTL* formula which is equivalent to ψ over every viable FDS.

4 Temporal Testers

In this section we present a construction of *temporal testers* [10] which are central to our verification process. Given a path formula φ, a tester $T[\varphi]$ for φ is an FDS with a distinguished boolean variable x_φ, such that $x_\varphi = 1$ at all positions in a $T[\varphi]$-computation in which φ is true. We often refer to the variable x_φ as the *output variable* of $T[\varphi]$. Temporal testers present a modular and symbolic version of the construction of a temporal tableau [14].

4.1 Testers for Basic Path Formulas

We first present a construction of testers for the basic path formulas. These will serve as the building blocks for the construction of a tester for a general path formula. We define a tester for each of the temporal operators $\bigcirc$, $\mathcal{U}$, $\mathcal{W}$, $\ominus$, $\circledcirc$, $\mathcal{S}$, and $\mathcal{B}$.

The set of variables of a tester $T[\varphi]$ for a basic path formula consists of the vocabulary of φ augmented by the variable x_φ.

The tester $T[\bigcirc p]$ for the basic path formula $\bigcirc p$ is defined as follows:

$$
T[\bigcirc p]: \quad
\begin{array}{ll}
V & : V_p \cup \{x_\bigcirc\} \\
\Theta & : \mathrm{T} \\
\rho & : x_\bigcirc = p' \\
\mathcal{J} & : \emptyset \\
\mathcal{C} & : \emptyset
\end{array}
$$

The tester $T[p\mathcal{U}q]$ for the basic path formula $p\mathcal{U}q$:

$$T[p\mathcal{U}q]: \quad \begin{array}{l} V : V_p \cup V_q \cup \{x_{\mathcal{U}}\} \\ \Theta : \mathrm{T} \\ \rho : x_{\mathcal{U}} = q \ \vee \ (p \ \wedge \ x'_{\mathcal{U}}) \\ \mathcal{J} : \{\neg x_{\mathcal{U}} \ \vee \ q\} \\ \mathcal{C} : \emptyset \end{array}$$

The tester $T[p\,\mathcal{W}\,q]$ for the basic path formula $p\,\mathcal{W}\,q$:

$$T[p\,\mathcal{W}\,q]: \quad \begin{array}{l} V : V_p \cup V_q \cup \{x_{\mathcal{W}}\} \\ \Theta : \mathrm{T} \\ \rho : x_{\mathcal{W}} = q \ \vee \ (p \ \wedge \ x'_{\mathcal{W}}) \\ \mathcal{J} : \{x_{\mathcal{W}} \ \vee \ (\neg p \ \wedge \ \neg q)\} \\ \mathcal{C} : \emptyset \end{array}$$

The tester $T[\ominus p]$ for the basic path formula $\ominus p$:

$$T[\ominus p]: \quad \begin{array}{l} V : V_p \cup \{x_{\ominus}\} \\ \Theta : x_{\ominus} = \mathrm{F} \\ \rho : x'_{\ominus} = p \\ \mathcal{J} : \emptyset \\ \mathcal{C} : \emptyset \end{array}$$

The tester $T[\circledcirc p]$ for the basic path formula $\circledcirc p$:

$$T[\circledcirc p]: \quad \begin{array}{l} V : V_p \cup \{x_{\circledcirc}\} \\ \Theta : x_{\circledcirc} = \mathrm{T} \\ \rho : x'_{\circledcirc} = p \\ \mathcal{J} : \emptyset \\ \mathcal{C} : \emptyset \end{array}$$

The tester $T[p\mathcal{S}q]$ for the basic path formula $p\mathcal{S}q$:

$$T[p\mathcal{S}q]: \quad \begin{array}{l} V \ : V_p \cup x_{\mathcal{S}} \\ \Theta \ : x_{\mathcal{S}} = q \\ \rho \ : x'_{\mathcal{S}} = q' \ \vee \ (p' \wedge x_{\mathcal{S}}) \\ \mathcal{J} \ : \emptyset \\ \mathcal{C} = : \emptyset \end{array}$$

The tester $T[p\,\mathcal{B}\,q]$ for the basic path formula $p\,\mathcal{B}\,q$:

$$T[p\,\mathcal{B}\,q]: \quad \begin{array}{l} V : V_p \cup x_{\mathcal{B}} \\ \Theta : x_{\mathcal{B}} = p \ \vee \ q \\ \rho : x'_{\mathcal{B}} = q' \ \vee \ (p' \wedge x_{\mathcal{B}}) \\ \mathcal{J} : \emptyset \\ \mathcal{C} : \emptyset \end{array}$$

Note that, in general, testers are not guaranteed to be deadlock-free. Consider, for example, a tester for the basic path formula $\Box\, p$. While we did not present such a tester in the preceding list, it can be derived from the tester for the formula $p\, \mathcal{W}\, \mathrm{F}$. This tester is given by:

$$
\begin{array}{lll}
 & V & : V_p \cup \{x_\Box\} \\
 & \Theta & : \mathrm{T} \\
T[\Box\, p]: & \rho & : x_\Box = p \ \wedge\ x'_\Box \\
 & \mathcal{J} & : \{x_\Box \ \vee\ \neg p\} \\
 & \mathcal{C} & : \emptyset
\end{array}
$$

This tester has the run

$$s_0 : \langle p : 0,\ x_\Box : 1 \rangle$$

which cannot be extended. Thus, it has some finite runs and is, therefore, not deadlock-free.

4.2 Testers for General LTL Formulas

Next, we present an incremental construction of a tester for a general LTL formula. First, we restrict our attention to LTL formulas whose principal operator is temporal (rather than boolean).

For an LTL formula ψ, we denote by $T[\psi]$ the temporal tester for ψ. Let $f(\varphi)$ be a principally temporal path formula containing one or more occurrences of the basic path formula φ. We denote by $f(x)$ the formula obtained from f by replacing all occurrences of φ by the boolean variable x. Then the construction principle is presented by the following recursive reduction formula:

$$T[f] \quad = \quad T[f(x_\varphi)] \ \|\|\ T[\varphi] \tag{1}$$

That is, we conjoin the tester for φ to the recursively constructed tester for the simpler formula $f(x_\varphi)$.
We illustrate this construction on the LTL formula $\Box \Diamond p$ for the case that p is a simple proposition (boolean variable). Application of the reduction principle leads to

$$T[\Box \Diamond p] \quad = \quad T[\Box\, x_\Diamond] \ \|\|\ T[\Diamond p]$$

Computing $T[\Diamond p]$ and $T[\Box\, x_\Diamond]$ separately and forming their synchronous parallel composition yields the following tester whose output variable is $x_\Box$.

$$
\begin{array}{lll}
 & V & : \{p, x_\Diamond, x_\Box\} \\
 & \Theta & : \mathrm{T} \\
T[\Box \Diamond p]: & \rho & : \left(x_\Diamond \ =\ p \ \vee\ x'_\Diamond\right) \ \wedge\ \left(x_\Box \ =\ x_\Diamond \ \wedge\ x'_\Box\right) \\
 & \mathcal{J} & : \{\neg x_\Diamond \ \vee\ p, \quad x_\Box \ \vee\ \neg x_\Diamond\} \\
 & \mathcal{C} & : \emptyset
\end{array}
$$

In general, for a principally temporal formula ψ, $T[\psi] = T_1 \,|||\, \cdots \,|||\, T_k$, where $T_1, \ldots, T_k$ are the temporal testers constructed for the principally temporal subformulas of ψ. $T[\psi]$ contains k auxiliary boolean variables, and the output variable of $T[\psi]$ is the output variable of T_1 — the last constructed tester.

In general, we carry the recursive reduction described by Equation (1) until we obtain the tester T_1 which is a tester for a basic path formula. We can carry it one step further and obtain an assertion which contains no further temporal operators. We refer to this assertion as the *redux* of the original LTL formula ψ, denoted by $\mathit{redux}(\psi)$. For the case that ψ is principally temporal, $\mathit{redux}(\psi)$ is the single output variable x_ψ. If we apply the recursive construction Equation (1) to an LTL formula which is not principally temporal, we may obtain a more complex assertion as the resulting redux.

Consider, for example, the LTL formula $\psi : \Box\, p \;\vee\; \Diamond\, q$. The corresponding tester is given by:

$$T[\psi] \quad = \quad T[\Box\, p] \,|||\, T[\Diamond\, q]$$

while $\mathit{redux}(\psi) = x_{\Box} \vee x_{\Diamond}$, where $x_{\Box}$ and $x_{\Diamond}$ are the output variables of $T[\Box\, p]$ and $T[\Diamond\, q]$, respectively.

5 Symbolic Model Checking CTL* Properties

In this section we present symbolic model checking of a CTL* formula over a finite-state FDS. The variables of both the FDS and the CTL* formula are restricted to finite domains.

Let $\mathcal{D}: \langle V, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$ be an FDS and $p = p(V)$ be an assertion over V. We define the *pre-image* of p in $\mathcal{D}$ to be the assertion

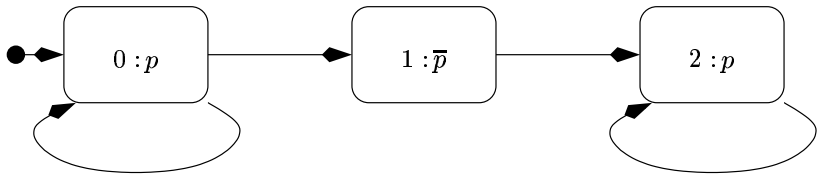
$$\rho \diamond p \quad = \quad \exists V' : \rho(V, V') \wedge p(V')$$

Every reachable state in $\mathcal{D}$ satisfying $\rho \diamond p$ has a successor satisfying p. Thus, $\rho \diamond p$ characterizes all the predecessors of p-states.
In a symmetric way, we define the *post-image* of an assertion $p = p(V)$ to be the assertion

$$p \diamond \rho \quad = \quad \exists V_0 : \rho(V_0, V) \wedge p(V_0)$$

The post-image $p \diamond \rho$ characterizes all the successors of p-states.

Example 1. Consider the following system:



The corresponding FDS is given by

$$\mathcal{D}: \begin{cases} V: \{\pi: [0..2],\ p: \textbf{boolean}\} \\ \Theta: \pi = 0 \ \wedge \ p \\ \rho: \begin{pmatrix} \pi = 0 \ \wedge \ (\pi' = 0 \ \wedge \ p' \ \vee \ \pi' = 1 \ \wedge \ \neg p') \ \vee \\ \pi \neq 0 \ \wedge \ (\pi' = 2 \ \wedge \ p') \end{pmatrix} \\ \mathcal{J} = \mathcal{C}: \emptyset \end{cases}$$

Computing the pre-image $\rho \diamond \neg p$, we obtain

$$\exists \pi', p': \rho \ \wedge \ \neg p' \quad = \quad (\pi = 0)$$

Thus, the set of states having a successor satisfying $\neg p$ includes the single state $\pi = 0$. ⌟

To capture the set of states that can reach a p-state in a finite number of ρ-steps, we define

$$\rho^* \diamond p \quad = \quad p \vee \rho \diamond p \vee \rho \diamond (\rho \diamond p) \vee \rho \diamond (\rho \diamond (\rho \diamond p)) \vee \cdots$$

In a similar way, we define

$$p \diamond \rho^* \quad = \quad p \vee p \diamond \rho \vee (p \diamond \rho) \diamond \rho \vee ((p \diamond \rho) \diamond \rho) \diamond \rho \vee \cdots$$

which capture the set of all states reachable in a finite number of ρ-steps from a p-state.

5.1 Statification

For a state formula φ, we denote by $\|\varphi\|$ the assertion characterizing the set of all $\mathcal{D}$-states satisfying φ. For the case that φ is an assertion, $\|\varphi\| = \varphi$. For the cases that we need to specify explicitly the system over which the formula is statified, we write $\|\varphi, \mathcal{D}\|$ to denote the statification of φ over FDS $\mathcal{D}$.

Claim 1 (Model Checking State Formulas).
For a state formula φ,

$$\mathcal{D} \models \varphi \quad \textit{iff} \quad \Theta \rightarrow \|\varphi\|.$$

It only remains to provide a recipe for the computation of $\|\varphi\|$ for the various state formulas.

As discussed in the introduction, our proof method is based on two decomposition principles as follows: A general state formula is decomposed into basic state formulas, and a general path formula is decomposed into basic path formulas. When these reductions are performed repeatedly, we remain with basic assertional formulas of the form $E_f p$ and $A_f p$ where p is an assertion. These assertional formulas are statified by the symbolic algorithm FEASIBLE presented in Subsection 5.4.2.

5.2 Eliminating Nested State Formulas

Let $f(\varphi)$ be a state formula containing one or more occurrences of the nested state formula φ, and let $q = \|\varphi\|$.

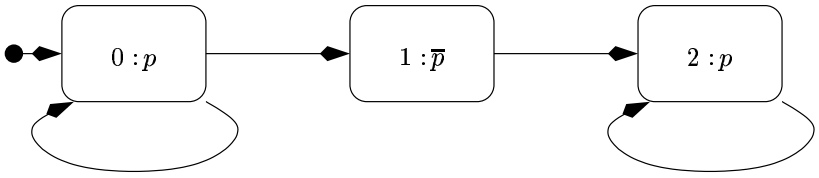
Claim 2 (Elimination of a nested state formulas).

$$\|f(\varphi)\| \quad = \quad \|f(q)\|,$$

where $f(q)$ is obtained by substituting q for all occurrences of φ in f.

Consider a general CTL* formula. Claim 2 enables us to eliminate all nested state formulas, starting with the innermost ones, until we end up with basic state formulas. Basic state formulas are statified by either temporal operator elimination (Subsection 5.4) or by statification of basic assertional formulas (Subsection 5.4.2).

Example 2. Consider the following system:



Assume we wish to model check the formula $f = A \diamondsuit \underbrace{A \square p}_{\varphi}$ over this system.

Following claim 2, we compute first

$$\|\varphi\| = \|A \square p\| = (\pi = 2).$$

Next, we compute

$$\|f(\|\varphi\|)\| = \|A \diamondsuit (\pi = 2)\| = (\pi > 0).$$

Now, it remains to check

$$\Theta \rightarrow \|f\| \quad = \quad (\pi = 0 \wedge p \ \rightarrow \ \pi > 0) \quad = \quad 0,$$

which shows that $A \diamondsuit A \square p$ does not hold on $\mathcal{D}$. ∎

5.3 Fair and Unfair Basic State Formula

A basic state formula φ is called a *fair state formula* if the single path quantifier appearing in φ is either E_f or A_f. Otherwise, it is called an *unfair state formula*. Given an unfair state formula φ, we denote by $fair(\varphi)$ the formula obtained from φ by replacing occurrences of E by E_f and occurrences of A by A_f.

Let $\mathcal{D} = \langle V, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$ be an FDS. We denote by $\mathcal{D}_{unfair}$ the FDS

$$\mathcal{D}_{unfair} : \langle V, \Theta, \rho, \emptyset, \emptyset \rangle$$

obtained from $\mathcal{D}$ by removing all justice and compassion requirements.
The following claim shows that, with no loss of generality, we can restrict our attention to fair basic state formulas.

Claim 3 (Sufficient to deal with Fair Basic State Formulas).
For an unfair basic state formula φ and deadlock-free FDS $\mathcal{D}$,

$$\mathcal{D} \models \varphi \quad \text{iff} \quad \mathcal{D}_{unfair} \models fair(\varphi)$$

5.4 Statification of Fair Basic State Formulas

In this subsection we show how to compute the statification of a fair basic state formula of the form $\mathcal{Q}_f\varphi$, where $\mathcal{Q}_f \in \{E_f, A_f\}$ is a fair path quantifier, and φ is an LTL formula. This is done in two steps: in 5.4.1 we show how to reduce a fair basic formula into a basic assertional formula, i.e., a formula of the form $\mathcal{Q}_f p$, where p is an assertion. Then, in 5.4.2 we show how to compute the statification of basic assertional formulas.

5.4.1 Reduction Into Basic Assertional Formulas

The modularity of CTL which enables us to model check a formula by successively computing $\|\varphi\|$ for all of its nested basic formulas has, for a long time, been considered a unique feature of CTL, and a major argument in the *branching* vs. *linear* debate (e.g. [12], [5]).

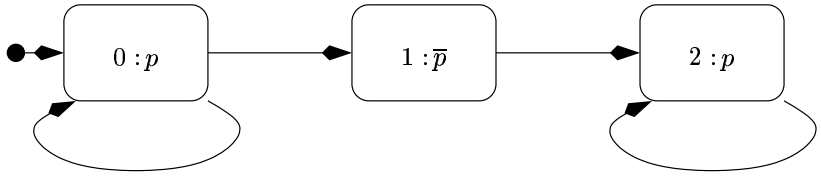
A similar modularity (though for a higher price) exists for the LTL component of a general basic state formula, as shown by the following:

Claim 4 (Elimination of Temporal Operators).
Let $\mathcal{Q}_f f(\psi)$ be a fair basic state formula containing one or more occurrences of the basic path formula ψ, where $\mathcal{Q}_f \in \{E_f, A_f\}$. Then, we can compute

$$\begin{array}{lcl} \|E_f f(\psi), \mathcal{D}\| & = & \exists x_\psi : (\|E_f f(x_\psi), \mathcal{D} \,|||\, T[\psi]\|) \\ \|A_f f(\psi), \mathcal{D}\| & = & \forall x_\psi : (\|A_f f(x_\psi), \mathcal{D} \,|||\, T[\psi]\|) \end{array}$$

where $T[\psi]$ is the temporal tester for ψ, x_ψ is the fresh variable introduced by $T[\psi]$. The expression $\|\mathcal{Q}_f f(x_\psi), \mathcal{D} \,|||\, T[\psi]\|$ stands for the statification of $\mathcal{Q}_f f(x_\psi)$ computed over the augmented FDS $\mathcal{D} \,|||\, T[\psi]$.

Example 3. Reconsider the system

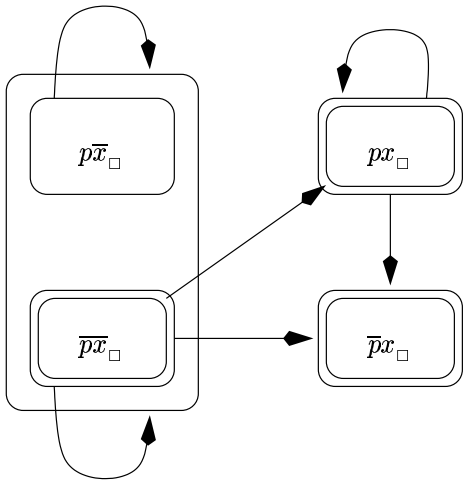


We wish to compute the statification $\|A_f \diamondsuit \square p\|$ over this system. Following the reductions of Claim 4, we obtain

$$\begin{array}{lcl} \|A_f \diamondsuit \square p, \mathcal{D}\| & = & \forall x_\square : \|A_f \diamondsuit x_\square, (\mathcal{D} \,|||\, T[\square p])\| \\ & = & \forall x_\diamondsuit, x_\square : \|A_f x_\diamondsuit, (\mathcal{D} \,|||\, T[\square p] \,|||\, T[\diamondsuit x_\square])\| \end{array}$$

First, we construct the tester $T[\square p]$.

$$T[\Box\, p]: \begin{cases} V : \{p, x_\Box\} \\ \Theta : \mathrm{T} \\ \rho : x_\Box \quad = \quad p \ \wedge \ x'_\Box \\ \mathcal{J} : x_\Box \ \vee \ \neg p \\ \mathcal{C} : \emptyset \end{cases}$$



The justice requirement $x_\Box \vee \neg p$ is intended to guarantee that we will not have a computation in which continuously $p = 1$, while $x_\Box = 0$.

Next, we form the parallel composition $\mathcal{D}_+ : \mathcal{D} \,|||\, T_\Box$ as presented in Fig. 1.

$\mathcal{D}_+ :$

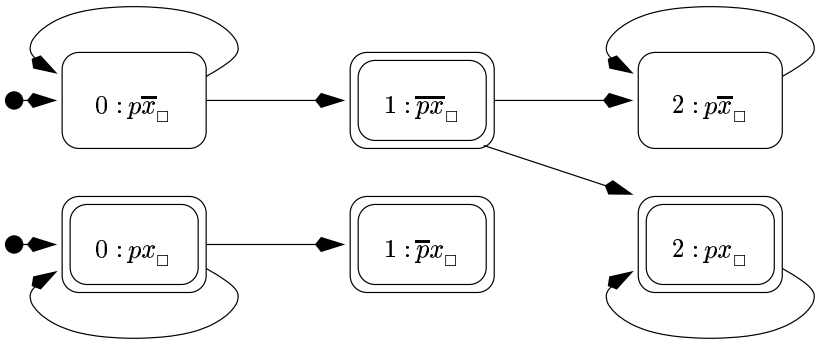


Fig. 1. System $\mathcal{D}_+$

Next, we construct the tester $T[\Diamond\, x_\Box]$ given by

$$T[\diamondsuit\, x_\Box] : \begin{cases} V : \{x_\Box, x_\diamond\} \\ \Theta : \mathrm{T} \\ \rho : x_\diamond \quad = \quad x_\Box \ \vee \ x'_\diamond \\ \mathcal{J} : x_\Box \ \vee \ \neg x_\diamond \\ \mathcal{C} : \emptyset \end{cases}$$

We form the parallel composition $\mathcal{D}_{++} = \mathcal{D}_+ \,|||\, T[\diamondsuit\, x_\Box] = \mathcal{D} \,|||\, T[\Box\, p] \,|||\, T[\diamondsuit\, x_\Box]$. This system is presented in Fig. 2. The justice requirements associated with $\mathcal{D}_{++}$ are

$$J_1 : x_\Box \ \vee \ \neg p, \qquad J_2 : \neg x_\diamond \ \vee \ x_\Box$$

In the diagram we denote the justice requirements satisfied by each state according to the following conventions: Every state is represented by a double oval. If the state satisfies J_1 then the inner oval is drawn in a "dash-dot" line style. Otherwise, the inner oval is drawn in a "solid" line style. Similarly, if the state satisfies J_2 then the outer oval is drawn in a "dash-dot" line style. Otherwise, the outer oval is drawn in a "solid" line style.

Using the methods of Sub-subsection 5.4.2, we can evaluate $\|A_f x_\diamond\|$ over $\mathcal{D}_{++}$, obtaining $\|A_f x_\diamond\| = 1$. We can therefore conclude that the original FDS $\mathcal{D}$ satisfies $A_f \diamondsuit \Box\, p$. ▟

A Summary Version of the Reduction

Consider a fair basic state formula $\mathcal{Q}_f\psi$. Systematic application of the reductions presented in Claim 4 performs successive augmentations of the FDS $\mathcal{D}$ by temporal testers corresponding to basic path sub-formulas of ψ, and successive replacements in ψ of these sub-formulas by the output variables of their corresponding testers.

Tracing these successive augmentations and replacements, we see that they are identical to the transformations used when computing the temporal tester of the LTL formula, as prescribed in Subsection 4.2. This observation leads to the following summary version of Claim 4 which, in one step, reduces a fair basic state formula into a basic assertional formula:

Claim 5 (From basic state formulas to basic assertional formulas). *Let $\mathcal{Q}_f\psi$ be a fair basic state formula. Then, we can compute*

$$\begin{array}{lcl} \|E_f\psi,\, \mathcal{D}\| & = & \exists \mathrm{vars}_\psi : (\|E_f \mathrm{redux}(\psi),\, \mathcal{D} \,|||\, T[\psi]\,\|) \\ \|A_f\psi,\, \mathcal{D}\| & = & \forall \mathrm{vars}_\psi : (\|A_f \mathrm{redux}(\psi),\, \mathcal{D} \,|||\, T[\psi]\,\|) \end{array}$$

where $T[\psi]$ is the temporal tester for ψ, and $\mathrm{redux}(\psi)$ *is the redux of ψ when constructing $T[\psi]$. The expression $\|\mathcal{Q}_f \mathrm{redux}(\psi),\, \mathcal{D} \,|||\, T[\psi]\,\|$ stands for the statification of $\mathcal{Q}_f \mathrm{redux}(\psi)$ computed over the augmented* FDS $\mathcal{D} \,|||\, T[\psi]$. *The variables* vars_ψ *are all the auxiliary variables introduced by the construction of $T[\psi]$.*

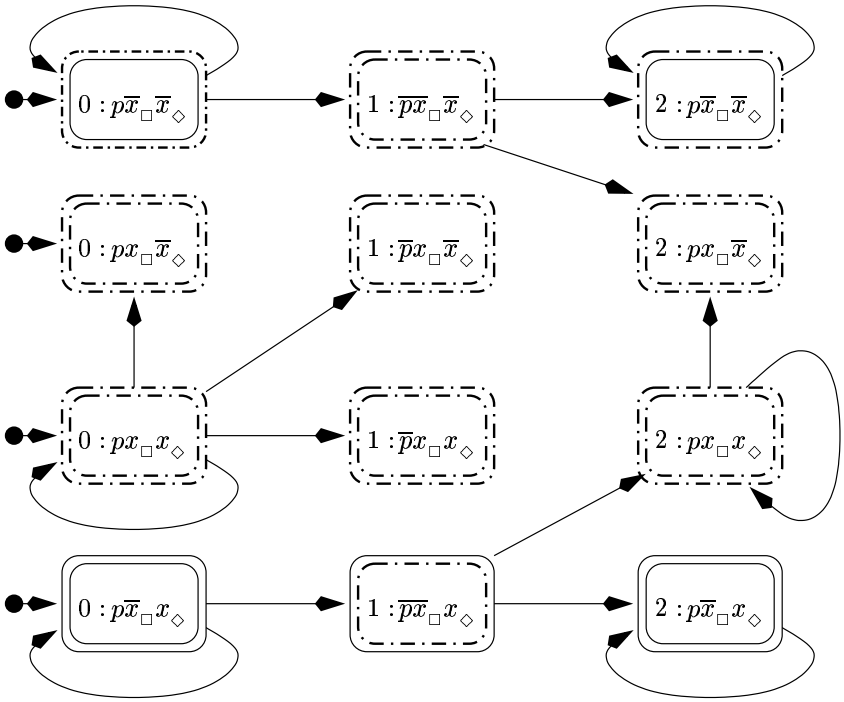


Fig. 2. System $\mathcal{D}_{++}$

This one-step reduction from basic state formulas to basic assertional formulas was first presented in [11].

Note that quantifying out the variables $vars_\psi$ guarantees that, when the computation is done, the statification $\|\mathcal{Q}_f\psi,\ \mathcal{D}\|$ is an assertion over $\mathcal{D}$. Thus, FDS's which are not deadlock-free arise only at intermediate stages in the statification of $\mathcal{Q}_f\psi$ but never at a stage where we may want to apply Claim 3 in order to convert an unfair path quantifier into a fair one.

Example 4. We can apply the reduction of Claim 5 in order to compute the statification $\|A_f \diamondsuit \square\, p\|$ over system $\mathcal{D}$ of Example 3. We start by computing the tester $T[\diamondsuit \square\, p]$ which is given by:

$$
T[\diamondsuit \square\, p]: \quad
\begin{array}{ll}
V & : \{p, x_{\square}, x_{\diamond}\} \\
\Theta & : \mathrm{T} \\
\rho & : \left(x_{\square} \;=\; p \;\wedge\; x'_{\square}\right) \;\wedge\; \left(x_{\diamond} \;=\; x_{\square} \;\vee\; x'_{\diamond}\right) \\
\mathcal{J} & : \{x_{\square} \;\vee\; \neg p, \quad \neg x_{\diamond} \;\vee\; x_{\square}\} \\
\mathcal{C} & : \emptyset
\end{array}
$$

The redux of $T[\diamondsuit\,\square\, p]$ is the output variable $x_{\diamondsuit}$. It only remains to form the synchronous parallel composition of $T[\diamondsuit\,\square\, p]$ with $\mathcal{D}$ which yields system $\mathcal{D}_{++}$, as presented in Fig. 2. Thus, Claim 5 yields the following reduction:

$$\|A_f \diamondsuit\,\square\, p, \mathcal{D}\| \quad = \quad \forall x_{\square}, x_{\diamondsuit} : Stat(A_f x_{\square}, \mathcal{D}_{++})$$

■

5.4.2 Statifying Basic Assertional Formulas

The statification of both types of basic assertional formulas, i.e., formulas of the form $E_f p$ and $A_f p$ for some assertion p, are based on the single FEASIBLE symbolic algorithm presented in Fig. 3.

Algorithm FEASIBLE($\mathcal{D}$)
$old := \text{F}$; $reach := \Theta \diamond \rho^*$; $new := reach$
while $(new \neq old)$ **do**
  $old := new$
  $new := new \ \wedge \ (\rho \diamond new)$
  – – Retain states which have a successor within new
  for each $J \in \mathcal{J}$ **do**
    $new := (new \ \wedge \ \rho)^* \diamond (new \ \wedge \ J)$
    – – Retain states with a new-path leading to a J-state
  for each $(p, q) \in \mathcal{C}$ **do**
    $new := \begin{pmatrix} new \ \wedge \ \neg p \\ \vee \ (new \ \wedge \ \rho)^* \diamond (new \ \wedge \ q) \end{pmatrix}$
    – – Retain states violating p or having a new-path
    – – leading to a q-state
return $reach \ \wedge \ \rho^* \diamond new$

Fig. 3. Algorithm FEASIBLE

The algorithm starts by placing in new the set of all reachable states. It then removes from this set all state which are obviously infeasible. For example, the assignment

$$new := new \ \wedge \ (\rho \diamond new)$$

removes from new all states which do not have a successor in new.

Algorithm FEASIBLE evaluates an assertion characterizing the set of all D-feasible states. Let U be the assertion evaluated by FEASIBLE ($\mathcal{D}$), and s be a state in $\mathcal{D}$.

Claim 6. *State s is $\mathcal{D}$-feasible iff $s \models U$.*

Proof: See [11].
The computation described in the algorithm of Fig. 3 can also be described more succinctly by the following fix-point formula:

$$\text{FEASIBLE} \quad = \quad \rho^* \diamond \ \nu\varphi \left(\begin{array}{l} \Theta \diamond \rho^* \ \wedge \ \rho \diamond \varphi \ \wedge \ \bigwedge_{J \in \mathcal{J}} (\varphi \wedge \rho)^* \diamond (\varphi \wedge J) \ \wedge \\ \bigwedge_{(p,q) \in \mathcal{C}} \neg p \ \vee \ (\varphi \wedge \rho)^* \diamond (\varphi \wedge q) \end{array} \right) \tag{2}$$

We can now evaluate the assertion characterizing the set of $\mathcal{D}$-states satisfying an existential and universal basic assertional formula, as follows:

$$\begin{array}{lcl} \|E_f p, \mathcal{D}\| & = & p \ \wedge \ \text{FEASIBLE}(\mathcal{D}) \\ \|A_f p, \mathcal{D}\| & = & \text{FEASIBLE}(\mathcal{D}) \rightarrow \ p \end{array}$$

Example 5.
Reconsider system $\mathcal{D}_{++}$ of Fig. 2 over which we wish to compute the statification $\|A_f x_\diamond\|$. Applying algorithm FEASIBLE, we obtain the following set of feasible states:

$$\{\langle 0 : p x_\Box x_\diamond \rangle, \ \langle 0 : p \overline{x}_\Box x_\diamond \rangle, \ \langle 1 : \overline{p x}_\Box x_\diamond \rangle, \ \langle 2 : p x_\Box x_\diamond \rangle\}$$

Since each of these states contains (and hence satisfies) $x_\diamond$, we conclude

$$\|A_f x_\diamond, \mathcal{D}_{++}\| \quad = \quad \mathit{feasible}(\mathcal{D}_{++}) \rightarrow x_\diamond \quad = \quad 1$$

implying that the the statification of $\|A_f x_\diamond\|$ over $\mathcal{D}_{++}$ contains all reachable states. From this, according to Example 3, we can conclude that system $\mathcal{D}$ satisfies $A_f \diamondsuit \Box p$. ▟

6 A Deductive Proof System for CTL* Properties

The deductive method can be used to verify CTL* properties over infinite-state systems. Recall our assumption that CTL* formulas are given in positive normal form. To simplify the presentation and get simpler proofs, we assume that the FDS representing the system is transformed into a JDS prior to the verification. We thus consider systems with no compassion requirements, and deal only with the *justice* requirements ($\mathcal{J}$).

Before proceeding, we remind the reader some of the previously introduced definitions:

- A *basic path formula* is a a path formula ψ whose principal operator is temporal, and such that ψ contains no additional temporal operators or path quantifiers.
- A *basic state formula* is a formula of the form $\mathcal{Q}\psi$, where $\mathcal{Q} \in \{E, A, E_f, A_f\}$ is a path quantifier and ψ is a path formula containing no additional path quantifier. In the case that $\mathcal{Q} \in \{E_f, A_f\}$, we refer to $\mathcal{Q}\psi$ also as a *fair basic state formula*.
- A *basic* CTL *state formula* is a basic state formula of the form $\mathcal{Q}\psi$ where ψ is a basic path formula.
- A *basic assertional formula* is a basic state formula of the form $\mathcal{Q}\psi$ where ψ is an assertion.

6.1 Decomposing a Formula into Basic State Formulas

Consider a CTL* formula φ which we wish to verify over an FDS $\mathcal{D}$. As a first step, we show how to reduce the task of verifying the formula φ into simpler subtasks, each required to verify a basic state formula over $\mathcal{D}$. This reduction repeatedly applies rule BASIC-STATE which is presented in Figure 4.

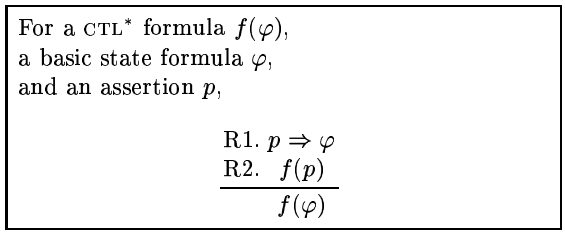


Fig. 4. BASIC-STATE.

The rule considers an arbitrary CTL* formula f which contains one or more occurrences of the basic state formula φ. The rule calls for an identification of an assertion p which under-approximates the set of states that satisfy the formula φ. It then reduces the task of verifying $f(\varphi)$ into the two simpler tasks of verifying the entailment $p \Rightarrow \varphi$, where φ is a basic state formula, and verifying the formula $f(p)$, obtained from f by substituting the assertion p for all occurrences of φ.

Claim 7 (Basic State). *Rule* BASIC-STATE *is sound and relatively complete for proving that* CTL* *formula f is valid over* FDS $\mathcal{D}$.

Proof Sketch: Relative completeness implies that, for every basic state formula φ, there exists an assertion p such that $p \Rightarrow \varphi$ is $\mathcal{D}$-valid. In fact, we claim that for every such φ, there exists an assertion, called the *statification* of φ and denoted by $\|\varphi\|$, such that $\|\varphi\|$ holds at precisely those states which satisfy φ. The notion of statification used here is identical to the one introduced in Section 5. The difference in the usage of this notion is that in Section 5 we provided a recipe showing how to compute $\|\varphi\|$ over a finite-state FDS using symbolic model checking techniques. Here we only claim that $\|\varphi\|$ can be expressed by an assertion, provided our underlying assertional language is expressive enough.

The proof of this fact proceeds by induction on the number of temporal operators within φ. We start with the case that φ is a basic CTL state formula. For the cases that the path quantifiers are unrestricted, these assertion can be expressed by a first-order formula, provided we can quantify over finite sequences of states. For example, the assertion expressing $\| E\Diamond r \|$ for an assertion r can be written as

$$\begin{aligned} \| E\Diamond r \| = \quad & \lambda s : \exists n \in \mathbf{N}, a : [1..n] \mapsto \Sigma : \\ & a[1] = s \ \wedge \ r(a[n]) \ \wedge \ \forall i : [1..n-1] : \rho(a[i], a[i+1]) \end{aligned} \tag{3}$$

This assertion claims the existence of a sequence of states $a[1..n]$, such that $a[1]$ coincides with s, each $a[i+1]$ is a ρ-successor of $a[i]$, and the final state $a[n]$ satisfies r.

For the case of fair path quantifiers such as A_f or E_f, we need to use fix-point operators as shown in Equation (2) in order to capture fair sequences.

Next, we consider the general case of a basic state formulas with more than a single temporal operator. According to Claim 3, it is sufficient to consider the case of fair basic state formulas of the form $\mathcal{Q}_f\psi$. We can then use the reduction techniques presented in Claim 4 by which

$$\begin{array}{lcl} \|E_f f(\psi),\, \mathcal{D}\,\| & = & \exists vars_\psi : (\|E_f f(x_\psi),\, \mathcal{D}\,|||\, T[\psi]\,\|) \\ \|A_f f(\psi),\, \mathcal{D}\,\| & = & \forall vars_\psi : (\|A_f f(x_\psi),\, \mathcal{D}\,|||\, T[\psi]\,\|) \end{array}$$

where $T[\psi]$ is the temporal tester for ψ, $vars_\psi$ are all the auxiliary variables introduced by the construction of $T[\psi]$, and $x_\psi \in vars_\psi$ is the principal output variable of $T[\psi]$. The computation of the statifications $\|E_f f(x_\psi),\, (\mathcal{D}\,|||\, T[\psi])\,\|$ and $\|A_f f(x_\psi),\, (\mathcal{D}\,|||\, T[\psi])\,\|$ can be expressed by the methods of Subsection 5.4.2. All of these computations can be expressed using first-order quantification over the state variables, and fix-points for the computation of FEASIBLE. Note that even though we are dealing here with systems with potentially infinitely many states, the structure of the temporal testers remains the same, and it is only required to add a single boolean variable as an auxiliary variable for each temporal operator. ⌟

Note that rule BASIC-STATE is actually sound (and relatively complete) for an arbitrary state formula. However, we recommend its use for the restricted case of basic state formulas. This recommendation suggests a systematic proof methodology by which one always deals with the innermost nested basic state formula, replacing it by an assertion, before continuing to deal with the rest of the formula.

Example 6.
Consider the system $\mathcal{D}$ presented in Fig. 5.

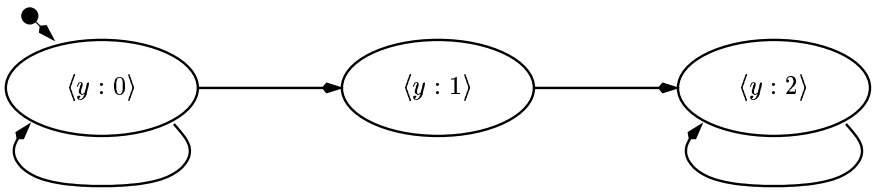


Fig. 5. An example system $\mathcal{D}$

This system has a single state variable y and no fairness conditions. For this system we wish to prove the property $f : E\,\Box\, E\Diamond(y = 1)$, claiming the existence of a run from each of whose states it is possible to reach a state at which $y = 1$.

Using rule BASIC-STATE, it is possible to reduce the task of verifying the non-basic formula $E \Box E \Diamond (y = 1)$ into the two tasks of verifying

R1. $(y = 0) \Rightarrow E \Diamond (y = 1)$
R2. $E \Box (y = 0)$

Note that, as the assertion p, we have chosen $y = 0$. The design of an appropriate assertion p which characterizes states satisfying φ is the part which requires creativity and ingenuity in the application of BASIC-STATE. ▟

In the rest of the section, we will present rules and methods which can be used to prove entailments of the form $p \Rightarrow \varphi$ for an assertion p and basic state formula φ.

6.2 Preliminary Inference Rules

We introduce two basic inference rules as part of the deductive system. Let $\mathcal{D}$ be an FDS and p, q be assertions. The first rule is *generalization*, presented in Figure 6. The rule transforms a state validity (denoted by $\mathbf{I}\models$) into a temporal validity (denoted by $\models$). The premise $\mathbf{I}\models p$ states that assertion p holds at every possible state, implying that p is a tautology. Then obviously, p holds at every reachable state of every model, and therefore the basic universal state formula $A \Box p$ holds at the initial state of every model (equivalently, $\mathcal{D} \models A \Box p$ for every FDS $\mathcal{D}$).

For an assertion p,

$$\frac{\mathbf{I}\models p}{\models A \Box p}$$

Fig. 6. GEN (generalization)

The second rule is *entailment modus ponens*, presented in Figure 7. The rule states that if every reachable state satisfies both p and the implication $p \rightarrow q$ (i.e., $\mathcal{D}$ satisfies $A \Box p$ and $A \Box (p \rightarrow q)$), then q holds at every reachable state ($\mathcal{D}$ satisfies $A \Box q$).

For assertions p and q,

$$\frac{\models A \Box p, \quad \models p \Rightarrow q}{\models A \Box q}$$

Fig. 7. EMP (entailment modus ponens)

Most of the rules in the proof system have the form $p \Rightarrow \varphi$, where p is an assertion. Rule INIT presented in Fig. 8 enables us to derive the cD-validity of the formula φ from the entailment $\Theta \Rightarrow \varphi$, where Θ is the initial condition of $\mathcal{D}$.

For CTL* formula φ,
and FDS $\mathcal{D}$ with initial condition Θ,

$$\frac{\models \Theta \Rightarrow \varphi}{\models \varphi}$$

Fig. 8. Rule INIT

In the following, we present a set of proof rules all having a common structure. Premises are stated above the line and the conclusion is presented below the line. Each rule claims that if the premises hold over $\mathcal{D}$, then so does the conclusion. When the validities of the premises and the conclusion are over different FDS's, the relevant FDS are stated explicitly, otherwise the common FDS is omitted.

6.3 Safety Rules

First we consider safety formulas of the form $\mathcal{Q} \Box q$, where q is an assertion and $\mathcal{Q} \in \{A, A_f, E, E_f\}$ is a path quantifier. We refer to such formulas as *invariance* formulas. We use the terms *universal* and *existential* invariance for the CTL* formulas $\{A \Box q, A_f \Box q\}$ and $\{E \Box q, E_f \Box q\}$ respectively.

6.3.1 Universal Invariance

In Figure 9, we present the rule for universal invariance, which is similar to the rule for LTL invariance. Rule A-INV states that if the set of premises I1 – I3

For an FDS $\mathcal{D}$ with transition relation ρ,
assertions p, q and φ,

$$\frac{\begin{array}{ll} \text{I1.} & p \Rightarrow \varphi \\ \text{I2.} & \varphi \Rightarrow q \\ \text{I3.} & \varphi \wedge \rho \Rightarrow \varphi' \end{array}}{p \Rightarrow A \Box q}$$

Fig. 9. A-INV (universal invariance)

are $\mathcal{D}$-valid, then the conclusion is $\mathcal{D}$-valid. Premise I1 and I2 are shorthand

notations for $\mathcal{D} \models A\,\Box(p \rightarrow \varphi)$ and $\mathcal{D} \models A\,\Box(\varphi \rightarrow q)$ respectively. Premise I1 states that every reachable p-state is also a φ-state. Similarly, premise I2 states that every reachable φ-state is a q-state. Assertion φ is introduced to strengthen assertion q in case q is not *inductive*, namely, in case q does not satisfy I3 (see [17] for a discussion on inductive assertions). Premise I3 is a shorthand notation for $\mathcal{D} \models A\,\Box(\varphi(V) \wedge \rho(V,V') \rightarrow \varphi(V'))$. The premise states that every ρ-successor of a reachable φ-state is a φ-state (equivalently, all transitions of $\mathcal{D}$ preserve φ). Rule A-INV establishes the invariance formula $A\,\Box\, q$ over all reachable p-states, for some assertion p.

Claim 8 (universal invariance). *Let $\mathcal{D}$ be an* FDS. *Rule* A-INV *is sound and relatively complete, for proving unrestricted universal (state) invariance over $\mathcal{D}$.*

Proof Sketch: The proof of completeness is based on the identification of an assertion φ, expressible in our assertional language, which satisfies the premises of rule A-INV. We follow the construction of [15] (further elaborated in [16]) to show the existence of an assertion characterizing all the states that can appear in a finite run of a system $\mathcal{D}$. ⌟

6.3.2 Existential Invariance We define a *well-founded domain* $(\mathcal{A}, \succ)$ to consist of a set $\mathcal{A}$ and a *well-founded order* relation $\succ$ on $\mathcal{A}$. The order relation $\succ$ is called *well-founded* if there does not exist an infinitely descending sequence $a_0, a_1, \ldots$ of elements of $\mathcal{A}$ such that $a_0 \succ a_1 \succ \ldots$

Next, we present three proof rules which together constitute a sound and (relatively) complete set for proving existential (state) invariance. The first rules, E-NEXT and E-UNTIL presented in Figures 10 and 11, prove the validity of the CTL* properties $E \bigcirc q$ and $qE\mathcal{U}r$ respectively, over all reachable p-states, where p, q and r are assertions. Both rules are defined for the unrestricted existential

For an FDS $\mathcal{D}$ with transition relation ρ,
assertions $p,\ q, \varphi$

$$\frac{p \;\Rightarrow\; \exists V' : \rho \wedge q'}{p \;\Rightarrow\; E \bigcirc q}$$

Fig. 10. E-NEXT.

path quantifier E which quantifies over any path, not necessarily a fair path (recall that E is weaker than E_f). While not being invariance rules by themselves, the E-NEXT and E-UNTIL rules are included in this subsection because they are essential for the invariance rule E_f-INV presented in Figure 12. The premise of rule E-NEXT is a shorthand notation for $\mathcal{D} \models A\,\Box(p(V) \rightarrow \exists V' : \rho(V,V') \wedge q(V'))$. The premise states that from every reachable p-state, there exists a $\mathcal{D}$-transition into a q-state.

For an FDS $\mathcal{D}$ with transition relation ρ,
assertions p, q, r and φ,
a well-founded domain $(\mathcal{A}, \succ)$, and a ranking function $\delta : \Sigma \mapsto \mathcal{A}$

$$\begin{array}{ll} \text{U1.} & p \Rightarrow \varphi \\ \text{U2.} & \varphi \Rightarrow r \vee (q \wedge \exists V' : (\rho \wedge \varphi' \wedge \delta' \prec \delta)) \\ \hline & p \Rightarrow q E \mathcal{U} r \end{array}$$

Fig. 11. E-UNTIL

The rule E-UNTIL uses a well-founded domain $(\mathcal{A}, \succ)$, and an intermediate assertion φ, associated with a ranking function δ. Function δ maps states into the set $\mathcal{A}$ and is intended to measure the distance of the current state to a state satisfying the goal r. The third rule, E_f-INV presented in Figure 12, is the existential invariance rule. We use the notation $(i \oplus_m 1)$ for $(i+1) mod\ m$. Rule E_f-INV proves a temporal property using three premises. Premises I1 and I2 use state reasoning, and premise I3 requires temporal reasoning. Premise I3 is resolved by the rules E-NEXT and E-UNTIL which transform the temporal reasoning into state reasoning. For the special case that $p = \Theta$ and $q = \mathrm{T}$, rule E_f-INV proves *feasibility* of $\mathcal{D}$. For the case that $p = q = \mathrm{T}$, the rule proves *viability* of $\mathcal{D}$.

For assertions $p, \varphi_0, \ldots, \varphi_m$,
an FDS $\mathcal{D}$ with justice requirements $J_0 = \mathrm{T}, J_1, \ldots, J_m \in \mathcal{J}$,

$$\begin{array}{lll} & \text{I1.} & p \Rightarrow \bigvee_{i=0}^{m} \varphi_i \\ \text{For } i = 0, \ldots, m, & & \\ & \text{I2.} & \varphi_i \Rightarrow J_i \\ & \text{I3.} & \varphi_i \Rightarrow q \wedge E \bigcirc (q E \mathcal{U} \varphi_{i \oplus_m 1}) \\ \hline & & p \Rightarrow E_f \Box q \end{array}$$

Fig. 12. E_f-INV.

Claim 9 (existential invariance). *Let $\mathcal{D}$ be an* FDS. *Rules* E-NEXT, E-UNTIL, *and* E_f-INV *are sound and relatively complete, for proving their respective conclusions.*

Proof Sketch: For rule E-NEXT, it is straightforward to write a first-order assertion φ which characterizes all the reachable states which have a successor satisfying p. For rule E-UNTIL, we can construct an assertion φ which characterizes all the states s from which there exists a q-path leading to an r-state. This

assertion is similar to the assertion described in Equation (3), except that we have to add the requirement that all states encoded in $a[1], \ldots, a[n-1]$ satisfy q. We can then use a ranking function δ which measures the length of the shortest q-path leading to an r-state. It only remains to show that these two constructs are expressible within our assertional language. For rule E_f-INV, we can use a maximal fix-point expression similar to Equation (2) to construct an assertion φ characterizing all accessible states initiating a continuous-q fair path. For the sub-assertions φ_i, we can take $\varphi \wedge J_i$. ▟

6.4 Liveness properties

6.4.1 Universal Liveness Under Justice In Figure 13, we present the rule for *universal eventuality* properties of the form $p \Rightarrow A_f \diamondsuit r$. The rule uses a well-founded domain $(\mathcal{A}, \succ)$, a ranking function δ, and a set of intermediate assertions $\varphi_1, \ldots, \varphi_m$. The function δ is intended to measure the distance of the current state to a state satisfying the goal q. Premise W1 states that every p-state satisfies q or one of $\varphi_1, \ldots, \varphi_m$. Premise W2 states that for every i, $1 \leq i \leq m$, a φ_i-state with rank $\delta = u$ is followed by either a q-state or a φ_i-state that does not satisfy J_i and has the same rank u, or by a φ_j-state ($1 \leq j \leq m$) with a smaller rank (i.e., $u \succ \delta$). The rule claims that if premise W1, and the set of m premises W2 are $\mathcal{D}$-valid, then the (fair) universal eventuality property $A_f \diamondsuit q$ is satisfied by all reachable p-states.

For an FDS $\mathcal{D}$ with transition relation ρ and justice set $\mathcal{J} = \{J_1, \ldots, J_m\}$,
assertions $p, q, \varphi_1, \ldots, \varphi_m$,
well-founded domain $(\mathcal{A}, \succ)$ and a ranking function $\delta : \Sigma \mapsto \mathcal{A}$

W1. $\quad p \quad \Rightarrow \quad q \vee \bigvee_{j=1}^{m} \varphi_j$

W2. For $i = 1, \ldots, m$

$$\varphi_i \wedge \rho \quad \Rightarrow \quad q' \vee (\neg J_i' \wedge \varphi_i' \wedge \delta = \delta') \vee \left(\bigvee_{j=1}^{m} \varphi_j' \wedge (\delta \succ \delta') \right)$$

$$p \Rightarrow A_f \diamondsuit q$$

Fig. 13. Rule A_f-EVENT (universal well-founded eventuality under justice).

Claim 10 (Universal eventuality). *Rule* A_f-EVENT *is sound and relatively complete, for proving the* $\mathcal{D}$*-validity of universal eventuality formulas.*

Proof Sketch: This rule is semantically equivalent to the LTL rule for the property $p \Rightarrow \diamondsuit q$. We refer the reader to [15] for the non-trivial proof of relative completeness of this rule.

▟

6.5 Basic Assertional Formulas

As the last rules for special cases, we present two rules dealing with the entailments $p \Rightarrow \mathcal{Q}q$, where p and q are assertions and $\mathcal{Q} \in \{A_f, E_f\}$ is a fair path quantifier.

Rule A_f-ASRT, presented in Figure 14, can be used in order to establish the validity of the entailment $p \Rightarrow A_f q$, for the case that p and q are assertions.

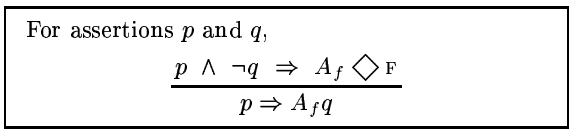


Fig. 14. A_f-ASRT.

The rule claims that, in order to prove the validity of $p \Rightarrow A_f q$, it is sufficient to show that no fair runs can depart from a reachable state satisfying $p \wedge \neg q$. This is shown by proving (using rule A_f-EVENT) that every fair run departing from a reachable $(p \wedge \neg q)$-state satisfies $A_f \Diamond \text{F}$ ("eventually false"), which is obviously impossible. Thus there can be no fair runs departing from such a state and, therefore, no such states are reachable as part of a computation.

The dual rule E_f-ASRT is presented in Figure 15.

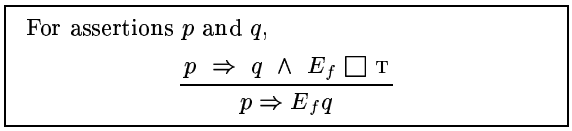


Fig. 15. E_f-ASRT.

This rule claims that, in order to prove $p \Rightarrow E_f q$, it is sufficient to show that every reachable p-state satisfies q and initiates at least one fair run.

6.6 Basic State Formulas

Finally, we present our general approach to the verification of an entailment of the form $p \Rightarrow \varphi$ for an assertion p and a basic state formula φ.

6.6.1 Unfair Basic State Formulas

The case that the basic state formula has the form $\varphi = \mathcal{Q}\psi$, where $\mathcal{Q} \in \{A, E\}$ is an unrestricted path quantifier, is treated in a way similar to Claim 3. That is,

For assertion p,
an unfair basic state formula $\mathcal{Q}\psi$,
and a deadlock-free FDS $\mathcal{D}$,

$$\frac{\mathcal{D}_{unfair} \models p \Rightarrow \mathcal{Q}_f\psi}{\mathcal{D} \models p \Rightarrow \mathcal{Q}\psi}$$

Fig. 16. Rule UNFAIR.

we convert unrestricted path quantifiers to their fair version. This is summarized in rule UNFAIR, presented in Fig. 16.
Consequently, it only remains to deal with formulas of the form $p \Rightarrow \mathcal{Q}_f\psi$.

6.6.2 Fair Basic State Formulas

The approach to the verification of a formula $p \Rightarrow \mathcal{Q}_f\psi$ is based on a successive elimination of temporal operators from the formula $\mathcal{Q}_f\psi$ until it becomes a basic assertional formula, to which we can apply rules A_f-ASRT or E_f-ASRT. Elimination of the temporal operators is based on the construction of temporal testers, as presented in Section 4.
Let $f(\varphi)$ be a fair basic state formula containing one or more occurrences of the path formula φ. In Fig. 17, we present the rule BASIC-PATH which reduces the proof of $f(\varphi)$ to the proof of $f(x_\varphi)$ over $\mathcal{D} \,|||\, T[\varphi]$, where $T[\varphi]$ is a temporal tester for φ, x_φ is the output variable of $T[\varphi]$, and $f(x_\varphi)$ is obtained from $f(\varphi)$ by replacing every occurrence of φ by x_φ.

For a basic state formula $f(\varphi)$,
a basic path formula φ,
assertion p, and an FDS $\mathcal{D}$,

$$\frac{\mathcal{D} \,|||\, T_\varphi \models p \Rightarrow f(x_\varphi)}{\mathcal{D} \models p \Rightarrow f(\varphi)}$$

Fig. 17. BASIC-PATH.

Example 7.
Reconsider system $\mathcal{D}$ first introduced in Fig. 5. For convenience, we duplicate it in Fig. 18. For this system we wish to prove the property $\text{T} \Rightarrow A_f \diamondsuit \Box \, even(y)$, stating that every computation eventually stabilizes with an even value of the state variable y.
Following are the first steps of the deductive proof for this property. The proof is presented in a goal-oriented style, where we identify for each goal the subgoals which are necessary in order to establish the goal and the rule which justifies

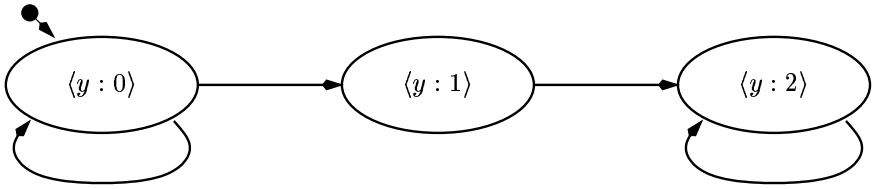


Fig. 18. An example system $\mathcal{D}$

the deductive step.

$$\begin{array}{lll} \mathcal{D} & \models & \text{T} \Rightarrow A_f \diamondsuit \square \, even(y) \\ \mathcal{D} \,|||\, T_\square & \models & \text{T} \Rightarrow A_f \diamondsuit x_\square \\ \mathcal{D} \,|||\, T_\square \,|||\, T_\diamondsuit & \models & \text{T} \Rightarrow A_f x_\diamondsuit \end{array} \qquad \begin{array}{l} \text{A tester for } x_\square = \square \, even(y) \\ \text{A tester for } x_\diamondsuit = \diamondsuit x_\square \end{array}$$

These proof steps reduce the task of verifying the formula $\text{T} \Rightarrow A_f \diamondsuit \square \, even(y)$ over system $\mathcal{D}$ to the verification of the simpler formula $\text{T} \Rightarrow A_f x_\diamondsuit$ over the system $\mathcal{D}^* = \mathcal{D} \,|||\, T_\square \,|||\, T_\diamondsuit$. The transition relation of system $\mathcal{D}^*$ is presented in Fig. 19.

The justice requirements associated with $\mathcal{D}^*$ are

$$J_1 : x_\square \vee \neg even(y), \qquad J_2 : \neg x_\diamondsuit \vee x_\square$$

We mark the states by double ovals, where the inner (respectively, outer) oval is drawn in a "dash-dot" line style iff the state satisfies J_1 (respectively, J_2).

We now use rule A_f-ASRT in order to reduce the goal $\mathcal{D}^* \models \text{T} \Rightarrow A_f x_\diamondsuit$ into the goal $\mathcal{D}^* \models \neg x_\diamondsuit \Rightarrow A_f \diamondsuit \text{F}$. This goal is proven using rule A_f-EVENT with the following choices:

$$\begin{array}{ll} p: & \neg x_\diamondsuit \\ q: & \text{F} \\ \delta: & 2 - y \\ \varphi_1: & \neg x_\diamondsuit \wedge \; even(y) \\ \varphi_2: & \neg x_\diamondsuit \wedge \neg even(y) \end{array}$$

6.7 Summary Version of Rule BASIC-PATH

Rule BASIC-PATH eliminates one temporal operator at a time. It is possible to reduce a fair basic state formula $\mathcal{Q}_f \psi$ into a basic assertional state formula in a single step, using the temporal tester for the entire LTL formula ψ. This is presented in rule LTL-REDUCE of Fig. 20.

7 Conclusions

In this paper, we presented a uniform approach to model checking and deductive verification of CTL* formulas over reactive systems. This approach is *compositional* in the structure of the verified formula. That is, the task of verifying a

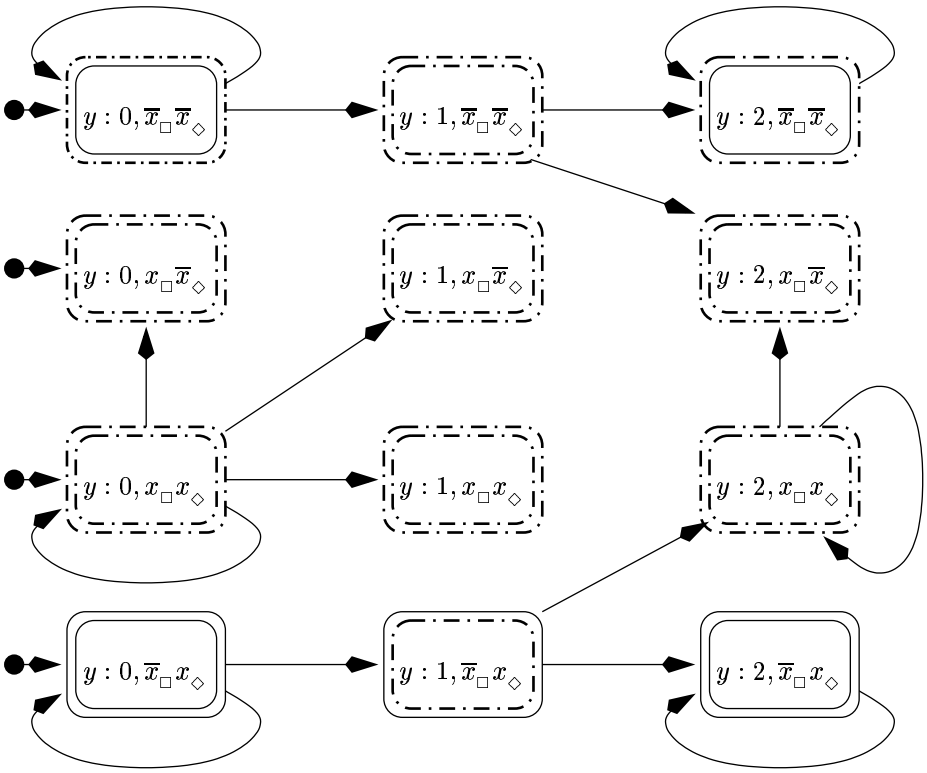


Fig. 19. The augmented system $\mathcal{D}^*$

complex CTL* formula φ over a system is broken into subtasks of verifying subformulas of φ. Modularity in the treatment of the LTL fragment of CTL* is based on the construction of *temporal testers* which introduces auxiliary boolean variables which are true whenever the LTL formula holds. This approach has close relation to [3] which encodes LTL fragments within CTL. The deductive system is sound and complete for verifying CTL* properties of reactive systems. The model-checking part has been implemented as reported in [11]. The deductive system is currently being implemented as a theory within PVS.

Acknowledgment

We gratefully acknowledge the continuous support, helpful suggestions, and critical assessment of the evolution of this work by Tamarah Arons who, in parallel with the theoretical developments reported here, implemented the CTL* deductive system as a PVS theory. These results will be reported in a separate publication.

For a basic state formula $\mathcal{Q}_f\psi$,
assertion p, and an FDS $\mathcal{D}$,

$$\frac{\mathcal{D} \,|||\, T_\psi \models p \Rightarrow \mathcal{Q}_f redux(\psi)}{\mathcal{D} \models p \Rightarrow \mathcal{Q}_f\psi}$$

Fig. 20. Rule LTL-REDUCE.

References

1. N. Bjørner, I. Browne, E. Chang, M. Colón, A. Kapur, Z. Manna, H. Sipma, and T. Uribe. STeP: The Stanford Temporal Prover, User's Manual. Technical Report STAN-CS-TR-95-1562, Computer Science Department, Stanford University, Nov. 1995.
2. Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *J. Comp. Systems Sci.*, 8:117–141, 1974.
3. E. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In D. L. Dill, editor, *Proc. 6th Conference on Computer Aided Verification*, volume 818 of *Lect. Notes in Comp. Sci.*, pages 415–427. Springer-Verlag, 1994.
4. E. Emerson. Temporal and modal logics. In J. van Leeuwen, editor, *Handbook of theoretical computer science*, volume B, pages 995–1072. Elsevier, 1990.
5. E. Emerson and J. Halpern. 'Sometimes' and 'not never' revisited: On branching time versus linear time. *J. ACM*, 33:151–178, 1986.
6. E. A. Emerson and C. Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8:275–306, 1987.
7. D. Gabbay. The declarative past and imperative future. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, volume 398 of *Lect. Notes in Comp. Sci.*, pages 407–448. Springer-Verlag, 1987.
8. Y. Kesten and A. Pnueli. Verification by augmented finitary abstraction. *Inf. and Comp.*, 163:203–243, 2000.
9. Y. Kesten and A. Pnueli. A deductive proof system for CTL*. In *13th International Conference on Concurrency Theory (CONCUR02)*, volume 2421 of *Lect. Notes in Comp. Sci.*, pages 24–39. Springer-Verlag, 2002.
10. Y. Kesten, A. Pnueli, and L. Raviv. Algorithmic verification of linear temporal logic specifications. In K. Larsen, S. Skyum, and G. Winskel, editors, *Proc. 25th Int. Colloq. Aut. Lang. Prog.*, volume 1443 of *Lect. Notes in Comp. Sci.*, pages 1–16. Springer-Verlag, 1998.
11. Y. Kesten, A. Pnueli, L. Raviv, and E. Shahar. Ltl model checking with strong fairness. *Formal Methods in System Design*, 2002. Accepted.
12. L. Lamport. 'sometimes' is sometimes 'not never': A tutorial on the temporal logic of programs. In *Proc. 7th ACM Symp. Princ. of Prog. Lang.*, pages 174–185, 1980.
13. D. Lehmann, A. Pnueli, and J. Stavi. Impartiality, justice and fairness: The ethics of concurrent termination. In *Proc. 8th Int. Colloq. Aut. Lang. Prog.*, volume 115 of *Lect. Notes in Comp. Sci.*, pages 264–277. Springer-Verlag, 1981.
14. O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symp. Princ. of Prog. Lang.*, pages 97–107, 1985.
15. Z. Manna and A. Pnueli. Completing the temporal picture. *Theor. Comp. Sci.*, 83(1):97–130, 1991.

16. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, New York, 1991.
17. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, New York, 1995.
18. K. Namjoshi. Certifying model checkers. In *G. Berry, H. Comon, and A. Finkel, editors,* Proc. 13^{th} Intl. Conference on Computer Aided Verification (CAV'01), *volume 2102 of* Lect. Notes in Comp. Sci., *Springer-Verlag*, 2001.
19. D. Peled, A. Pnueli, and L. Zuck. From falsification to verification. In *FTTCS*, volume 2245 of *Lect. Notes in Comp. Sci.*, pages 292–304. Springer-Verlag, 2001.
20. A. Pnueli and Y. Kesten. *Models, Algebras and Logic of Engineering Software*, chapter Algorithmic and Deductive Verification Methods for CTL*. NATO Science Series: Computer and Systems Science. IOS Press, 2002. To Appear.
21. A. Pnueli and R. Rosner. A framework for the synthesis of reactive modules. In F. Vogt, editor, *Proc. Intl. Conf. on Concurrency: Concurrency 88*, volume 335 of *Lect. Notes in Comp. Sci.*, pages 4–17. Springer-Verlag, 1988.
22. M. Reynolds. An axiomatization of full computation tree logic. *J. Symb. Logic*, 66(3):1011–1057, 2001.
23. H. Sipma, T. Uribe, and Z. Manna. Deductive model checking. *Formal Methods in System Design*, 15(1):49–74, 1999.
24. C. Sprenger. *On the Verification of CTL* Properties of Infinite-State Reactive Systems.* PhD thesis, Swiss Federal Institute of Technology, Lausanne, 2000.
25. F. Stomp, W.-P. de Roever, and R. Gerth. The μ-calculus as an assertion language for fairness arguments. *Inf. and Comp.*, 82:278–322, 1989.
26. M. Y. Vardi. Verification of concurrent programs – the automata-theoretic framework. *Annals of Pure Applied Logic*, 51:79–98, 1991.