A Deductive Proof System for CTL*⋆

Amir Pnueli[1] and Yonit Kesten[2]

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

Abstract. The paper presents a sound and (relatively) complete deductive proof system for the verification of CTL* properties over possibly infinite-state reactive systems. The proof system is based on a set of proof rules for the verification of *basic* CTL* formulas, namely CTL* formulas with no embedded path quantifiers. We first present proof rules for some of the most useful basic CTL* formulas, then present a methodology for transforming an arbitrary basic formula into one of these special cases. Finally, we present a rule for decomposing the proof of a general (non-basic) CTL* formula into proofs of basic CTL* formulas.

1 Introduction

The paper presents a sound and (relatively) complete deductive proof system for the verification of CTL* properties over possibly infinite-state reactive 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 both LTL and CTL as sub-logics. A complete deductive proof system for linear-time temporal logic (LTL) has been presented in [16] and further elaborated in [17] and [18]. This proof system has been successfully implemented in the Stanford Temporal Verifier STeP [15]. The presented work can be viewed as an extension of the approach of [16] 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.

⋆ This research was supported in part by the John von Newman Minerva Center for the Verification of Reactive 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 [19, 20].

The proof system presented here is based on a set of proof rules for *basic* CTL* formulas, which are CTL* formulas with no embedded path quantifiers. Thus, a basic CTL* formula has the form $\mathcal{Q}\varphi$, where $\mathcal{Q}$ is a path quantifier and φ is a *path formula* (according to the CTL* terminology) or, equivalently, can be described as an LTL formula. As a first step, we reduce the problem of verifying an arbitrary CTL* formula into a set of verification tasks of the form $p_i \Rightarrow \beta_i$ where p_i is an assertion and β_i is a basic CTL* 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 CTL* 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 a 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 [5]. It has been used again in [12] for the construction of a symbolic model checker (SMC) for CTL* properties over finite state systems.

Concentrating on rules for verifying properties of the form $p \Rightarrow \beta$, we present first a set of rules for the special cases of basic formulas of the forms $A\Box\, q$, $E\bigcirc q$, $q\, E\mathcal{U}\, r$, $E_f\Box\, q$, and $A_f \diamondsuit q$. For the universal path quantifiers, these rules are adapted versions of corresponding LTL rules taken from [16].

To deal with *arbitrary* basic CTL* 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* [11]. A basic path formula is a path formula whose principal operator is temporal and it contains no other nested temporal operators.

Thus, our proof method is based on two *statification* transformations which successively replace temporal formulas by assertions which contain no path quan-

tifiers or temporal operators. The first transformation replaces a basic CTL* 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 this proof system with the LTL deductive proof system presented in [16] and elaborated in [17, 18, 15]. Similar to the approach presented here, 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, [16] 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., [6] 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, based on successive elimination of temporal operators, can be viewed as an incremental tableau construction and offers a viable new approach to the verification of arbitrary LTL formulas, even though it is presented as a 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 [18] and provides realistic tools for verifying non-trivial reactive systems, such as those implemented in the STeP system [15]. 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 complex-

ity 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 [25]. 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*. In Section 4, we show how to decompose the task of verifying a general CTL* formula into tasks, each verifying a basic CTL* formula. In Section 5, we present the methodology by which we propose to prove a basic CTL* formula. This methodology starts by presenting a set of proof rules for some of the most useful basic CTL* properties, and claim soundness and completeness of these rules. We then show how to reduce an arbitrary basic CTL* formula to one of these special cases. Finally, in Section 6, we present an example of the application of these rules.

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. Let $\mathcal{D}$ be an FDS for which the above components have been identified. We define a *run* of $\mathcal{D}$ to be a finite or infinite sequence of states $\sigma : s_0, s_1, ...,$ satisfying the requirement of

- *Consecution:* For each $j = 0, 1, ...,$ the state s_{j+1} is a $\mathcal{D}$-successor of the state s_j.

and such that it is either infinite, or terminates at a state s_k which has no $\mathcal{D}$-successors.

We denote by $runs(\mathcal{D})$ the set of runs of $\mathcal{D}$. An infinite run of $\mathcal{D}$ is called *fair* 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 say that a fair run $\sigma : s_0, s_1, \ldots$ is a *computation* of $\mathcal{D}$ if it satisfies

- *Initiality:* s_0 is initial, i.e., $s_0 \models \Theta$.

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

A state s is said to be $\mathcal{D}$-*reachable* if there exists a run $s_0, s_1, s_2, \ldots, s_k = s, \ldots$ in $\mathcal{D}$ such that $s_0 \models \Theta$. Let p be an assertion, and s be a state in $\mathcal{D}$. We say that s is $p_{\mathcal{D}}$-*reachable* if s is reachable from a $\mathcal{D}$-reachable p-state. We say that a state s is $\mathcal{D}$-*feasible* if it participates in some computation of $\mathcal{D}$. 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 $\mathcal{D}$-reachable state is $\mathcal{D}$-feasible. Note that the FDS model does not guarantee viability.

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 [KP00]). 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 LTL formula φ, as described in section 5. We define the *synchronous* parallel

composition of two FDS's to be

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

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

3 Branching Temporal Logic

In the following we define the branching temporal logic CTL* (see [4]). 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. A CTL* formula is constructed out of *assertions* (formulas over $\mathcal{L}$) to which we apply the boolean operators, temporal operators and path quantifiers. The basic temporal operators are

$\bigcirc$ – Next $\quad \mathcal{U}$ – Until $\quad \mathcal{W}$ – waiting-for

Additional temporal operators may be defined as follows:

$$\Diamond p = \mathrm{T} \mathcal{U} p, \quad \Box p = \neg \Diamond \neg p,$$

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 tree generated by an FDS. We use the terms *path* and *fair path* as synonymous to a *run* and a *fair run* respectively, over an FDS. Let $\pi : s_0, s_1, \ldots$ be a run of $\mathcal{D}$. Then, we write $\pi[0]$ to denote s_0, the first state in π and, for $j \geq 0$, we write $\pi[j..] = s_j, s_{j+1}, \ldots$ to denote the suffix of π obtained by omitting the first j states. If the path π is finite, we use $|\pi|$ to denote its length.

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, $E_f p$ and $A_f p$ are state formulas.
- If p and q are state formulas then so are $p \vee q$ and $p \wedge q$.

Path formulas:

[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 [16], based on [26].

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

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

We say that p is a *basic* CTL* *formula* if p is a CTL* formula with no embedded path quantifiers. A basic CTL* formula of the form $A\psi$ or $A_f\psi$ ($E\psi$ or $E_f\psi$) is called a basic *universal* (*existential*) 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. Note that the set of basic universal CTL* formulas corresponds to the set of linear temporal logic formulas (LTL). 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,
  $(\mathcal{D}, s) \models Ep \quad \Leftrightarrow \quad (\mathcal{D}, \pi) \models p$ for some path $\pi \in \mathit{runs}(\mathcal{D})$ satisfying $\pi[0] = s$.
  $(\mathcal{D}, s) \models Ap \quad \Leftrightarrow \quad (\mathcal{D}, \pi) \models p$ for all paths $\pi \in \mathit{runs}(\mathcal{D})$ satisfying $\pi[0] = s$.

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

Path formulas are interpreted over runs of $\mathcal{D}$. We define the notion of a CTL* formula p holding at a run $\pi \in \mathit{runs}(\mathcal{D})$, denoted $(\mathcal{D}, \pi) \models p$, as follows:

- For a state formula p,
  $(\mathcal{D}, \pi) \models p \quad \Leftrightarrow \quad (\mathcal{D}, \pi[0]) \models p.$
- For path formulas p and q,
  $(\mathcal{D}, \pi) \models p \vee q \quad \Leftrightarrow \quad (\mathcal{D}, \pi) \models p$ or $(\mathcal{D}, \pi) \models q$
  $(\mathcal{D}, \pi) \models p \wedge q \quad \Leftrightarrow \quad (\mathcal{D}, \pi) \models p$ and $(\mathcal{D}, \pi) \models q$
  $(\mathcal{D}, \pi) \models \bigcirc p \quad \Leftrightarrow \quad |\pi| > 1$ and $(\mathcal{D}, \pi[1..]) \models p$
  $(\mathcal{D}, \pi, j) \models p \mathcal{U} q \quad \Leftrightarrow \quad (\mathcal{D}, \pi[k..]) \models q$ for some $k \geq 0$, and $(\mathcal{D}, \pi[i..]) \models p$ for every i, $0 \leq i < k$
  $(\mathcal{D}, \pi, j) \models p \mathcal{W} q \quad \Leftrightarrow \quad (\mathcal{D}, \pi) \models p \mathcal{U} q$, or $(\mathcal{D}, \pi[i..]) \models p$ for all $i < |\pi|$.

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 $\mathcal{D}$-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'.

Without loss of generality, we assume that a formula is given in *positive normal form*, which means that negation is only applied to assertions, namely, to formulas with no temporal or path operators. Any CTL* formula can be brought to a congruent positive form by a repeated application of the following rewriting rules.

$$
\begin{array}{ll}
\neg\neg p & \rightarrow p \\
\neg(p \wedge q) & \rightarrow (\neg p \vee \neg q) \\
\neg(p \vee q) & \rightarrow (\neg p \wedge \neg q) \\
\neg \bigcirc p & \rightarrow \bigcirc \neg p \\
\neg(p \mathcal{U} q) & \rightarrow (\neg q)\, \mathcal{W} (\neg p \wedge \neg q) \\
\neg(p \mathcal{W} q) & \rightarrow (\neg q)\, \mathcal{U} (\neg p \wedge \neg q) \\
\neg A_f p & \rightarrow E_f \neg p \\
\neg E_f p & \rightarrow A_f \neg p
\end{array}
$$

4 Decomposing a Formula into Basic CTL* 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 CTL* formula over $\mathcal{D}$. This reduction repeatedly applies rule BASIC-STATE which is presented in Fig. 1.

For a formula $f(\varphi)$,
a basic CTL* formula φ,
and an assertion p,

$$\frac{\text{R1. } p \Rightarrow \varphi \qquad \text{R2. } f(p)}{f(\varphi)}$$

Fig. 1. BASIC-STATE.

The rule considers an arbitrary formula f which contains one or more occurrences of the basic CTL* formula φ. The rule calls for an identification of an assertion p which characterizes all states which 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 CTL* formula, and the formula $f(p)$ obtained from f by substituting the assertion p for all occurrences of φ.

Example

Consider the system $\mathcal{D}$ presented in Fig. 2.

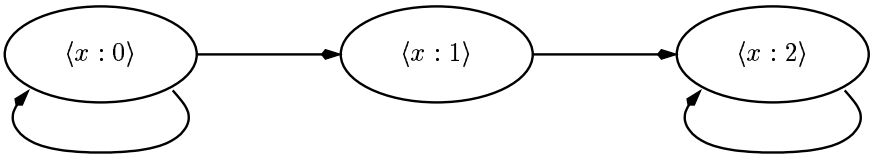

Fig. 2. An example system $\mathcal{D}$

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

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

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

Note that, as the assertion p, we have chosen $x = 0$. 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 following section, we present methods which can be used to prove entailments of the form $p \Rightarrow \varphi$ for an assertion p and basic CTL* formula φ.

5 Proof Rules for Basic CTL* Properties

In this section we present a set of proof rules for proving entailments of the form $p \Rightarrow \varphi$ for an assertion p and a basic CTL* formula φ. To simplify the presentation, we consider only systems with no compassion requirements, and deal only with the *justice* requirements ($\mathcal{J}$).

5.1 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 3. The rule transforms a state validity (denoted by $\models$) into a temporal validity (denoted by $\models$). The premise $\models p$ states that assertion p holds at every possible state. Then obviously, p holds at every reachable state of every model, and therefore the basic universal CTL* 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,

$$\frac{\models p}{\models A\,\Box\, p}$$

Fig. 3. GEN (generalization)

The second rule is *entailment modus ponens*, presented in Figure 4. 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. 4. EMP (entailment modus ponens)

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.

5.2 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.

Universal Invariance In Figure 5, we present the rule for universal invariance, which is similar to the rule for LTL invariance. Rule A-INV states that if the set

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. 5. A-INV (universal invariance)

of premises I1 – I3 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 $\mathcal{D}$-reachable p-state is also a φ-state. Similarly, premise I2 states that every $\mathcal{D}$-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 [18] 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 $\mathcal{D}$-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 $p_{\mathcal{D}}$-reachable states (i.e., all states that are reachable from a $\mathcal{D}$-reachable p-state), for some assertion p.

Claim (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 [16] (further elaborated in [17]) to show the existence of an assertion characterizing all the states that can appear in a finite run of a system $\mathcal{D}$. □

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 6 and 7, prove the validity of the CTL* properties $E \bigcirc q$ and $qE\mathcal{U}r$ respectively, over all $\mathcal{D}$-reachable p-states, where p, q and r are assertions. Both rules are defined for the unrestricted

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

$$\begin{array}{ll} \text{N1.} & p \Rightarrow \varphi \\ \text{N2.} & \varphi \Rightarrow \exists V' : \rho \wedge q' \\ \hline & p \Rightarrow E \bigcirc q \end{array}$$

Fig. 6. E-NEXT.

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 $\delta : \Sigma \mapsto \mathcal{A}$

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

Fig. 7. E-UNTIL

existential 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 8. Rule E-NEXT uses an intermediate assertion φ which strengthen assertion p in case p is not inductive. Premise N2 is a shorthand notation for $\mathcal{D} \models A\,\Box(\varphi(V) \rightarrow \exists V' : \rho(V, V') \wedge q(V'))$. The premise states that from every $\mathcal{D}$-reachable φ-state, there exists a $\mathcal{D}$-transition into a q-state.

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 8, 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 $q = \text{T}$, rule E_f-INV proves *feasibility* of $\mathcal{D}$. For the case that $p = q = \text{T}$, the rule proves *viability* of $\mathcal{D}$.

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

I1. $p \Rightarrow \bigvee_{i=0}^{m} \varphi_i$

For $i = 0, \ldots, m$,

I2. $\varphi_i \Rightarrow J_i$

I3. $\varphi_i \Rightarrow q \wedge E \bigcirc (q E \mathcal{U} \varphi_{i \oplus_m 1})$

$p \Rightarrow E_f \Box q$

Fig. 8. E_f-INV.

Claim (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 $\mathcal{D}$-reachable p-states, i.e. all p-states participating in a run of $\mathcal{D}$. For rule E-UNTIL, we can use again an assertion φ which characterizes all the $\mathcal{D}$-reachable p-states. We can use a ranking function δ which measures the number of steps from a $\mathcal{D}$-reachable p-state s to its closest r-state reachable by a continuous q-path. 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 to construct an assertion φ characterizing all accessible states initiating a continuous-q fair path. For the sub-assertions φ_i, we can take $\varphi \wedge J_i$. □

5.3 Liveness properties

Universal Liveness Under Justice In Figure 9, we present the rule for *universal eventuality* properties of the form $p \Rightarrow A_f \diamondsuit r$. The rule uses a well-founded domain $(\mathcal{A}, \succ)$, and a set of intermediate assertions $\varphi_1, \ldots, \varphi_m$, each associated with its own ranking function δ_i. Each function δ_i 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 $\delta_i = u$ 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., $u \succ \delta_j$). 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 $\mathcal{D}$-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 ranking functions $\delta_1, \ldots, \delta_m : \Sigma \mapsto \mathcal{A}$

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

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

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

$$p \Rightarrow A_f \diamondsuit q$$

Fig. 9. A_f-EVENT (universal well-founded eventuality under justice).

Claim (Completeness of 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 [16] for the non-trivial proof of relative completeness of this rule.

□

5.4 Assertional Basic CTL* 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 Fig. 10, 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.

For assertions p and q,

$$\frac{p \ \wedge \ \neg q \ \Rightarrow \ A_f \diamondsuit \mathrm{F}}{p \Rightarrow A_f q}$$

Fig. 10. 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 $A_f \diamondsuit \mathrm{F}$ ("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 Fig. 11.
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.

For assertions p and q,

$$\frac{p \Rightarrow q \wedge E_f \Box \mathrm{T}}{p \Rightarrow E_f q}$$

Fig. 11. E_f-ASRT.

5.5 Basic CTL* Properties: The General Case

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

The approach is based on a successive elimination of temporal operators from the formula φ until it becomes an assertional basic CTL* 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 follows.

We define a temporal tester for each of the three temporal operators $\bigcirc$, $\mathcal{U}$ and $\mathcal{W}$. These testers are a simplified version of the tableau construction presented in [14] and its later symbolic version which was referred to as *tester* [11]. For an assertion p, we denote by V_p the set of variables occuring in p.
For the basic path formula $\bigcirc p$, we construct the tester $T_{\bigcirc p}$ as follows:

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

For the basic path formula $p \mathcal{U} q$, we construct the tester $T_{p\mathcal{U}q}$ as follows:

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

For the basic path formula $p \mathcal{W} q$, we construct the tester $T_{p\mathcal{W}q}$ as follows:

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

Let $f(\varphi)$ be an arbitrary CTL* formula containing one or more occurrences of the basic path formula φ. In Fig. 12, we present the rule BASIC-PATH which reduces the proof of $f(\varphi)$ to the proof of $f(x_\varphi)$, where x_φ is a boolean variable, and $f(x_\varphi)$ is obtained from $f(\varphi)$ by replacing every occurrence of φ by x_φ.

6 An Example

In this section, we present an example system and a deductive proof of a property it has. In Fig. 13, we present the system $\mathcal{D}$. This system has a single state variable

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

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

Fig. 12. BASIC-PATH.

x and no fairness conditions. For this system we wish to prove the property $\mathrm{T} \Rightarrow A_f \diamondsuit \square \, even(x)$, stating that every computation eventually stabilizes with an even value of the state variable x.

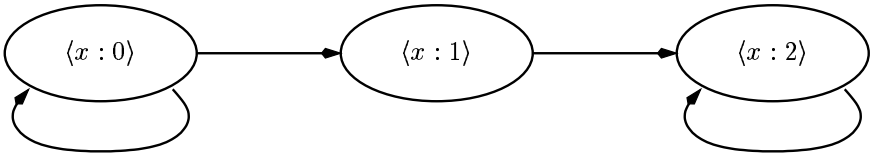

Fig. 13. An example system $\mathcal{D}$

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 the deductive step.

$\mathcal{D}$	$\models$	$\mathrm{T} \Rightarrow A_f \diamondsuit \square \, even(x)$	A tester for $x_\square = \square \, even(x)$
$\mathcal{D} \,\|\|\| \, T_\square$	$\models$	$\mathrm{T} \Rightarrow A_f \diamondsuit x_\square$	A tester for $x_\diamondsuit = \diamondsuit x_\square$
$\mathcal{D} \,\|\|\| \, T_\square \,\|\|\| \, T_\diamondsuit$	$\models$	$\mathrm{T} \Rightarrow A_f x_\diamondsuit$	

Thus, these proof steps reduced the task of verifying the formula $\mathrm{T} \Rightarrow A_f \diamondsuit \square \, even(x)$ over system $\mathcal{D}$ to the verification of the simpler formula $\mathrm{T} \Rightarrow A_f x_\diamondsuit$ over the system $\mathcal{D}^* = \mathcal{D} \,|||\, T_\square \,|||\, T_\diamondsuit$. The transition relation of system $\mathcal{D}^*$ is presented in Fig. 14.

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

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

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

p : $\neg x_\diamondsuit$	
q : F	
φ_1 : $\neg x_\square \wedge \neg x_\diamondsuit \wedge even(x)$	$\delta_1 : 2 - x$
φ_2 : $\neg x_\square \wedge \neg x_\diamondsuit \wedge \neg even(x)$	$\delta_2 : 2 - x$

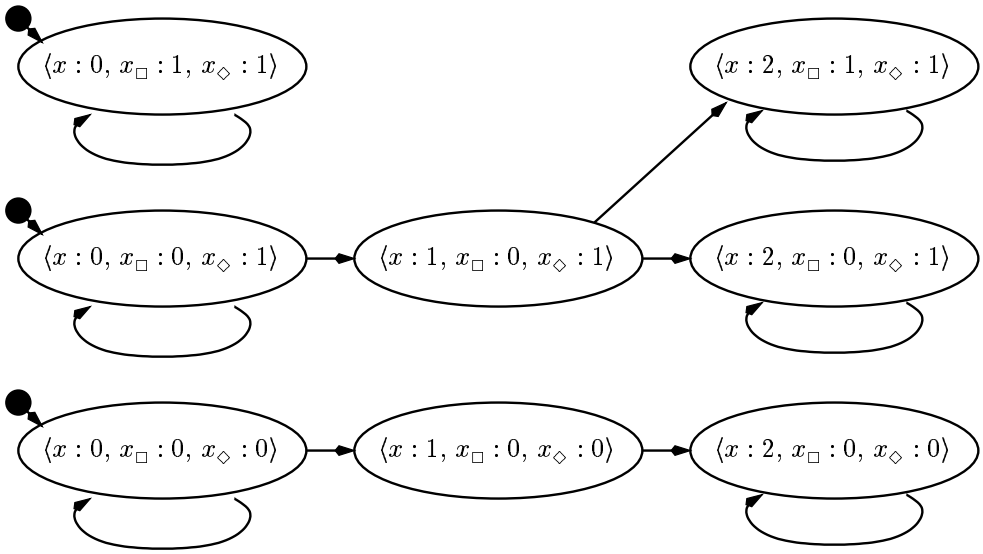

Fig. 14. The augmented system $\mathcal{D}^*$

References

1. E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM Trans. Prog. Lang. Sys.*, 8:244–263, 1986.
2. E. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. *CAV'94*, LNCS 818, pages 415–427.
3. M. Daniele, F. Giunchiglia, and M. Y. Vardi. Improved automata generation for linear time temporal logic. *CAV'99*, LNCS 1633, pages 255–265.
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 C. Lei. Modalities for model checking: Branching time strikes back. *POPL'85*, pages 84–96.
6. 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.
7. P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. *CAV'01*, LNCS 2102.
8. R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. *PSTV'95*, pages 3–18.
9. Y. Kesten, Z. Manna, H. McGuire, and A. Pnueli. A decision algorithm for full propositional temporal logic. *CAV'93*, pages 97–109.
10. Y. Kesten and A. Pnueli. Verification by augmented finitary abstraction. *Information and Computation, a special issue on Compositionality*, 163:203–243, 2000.
11. Y. Kesten, A. Pnueli, and L. Raviv. Algorithmic verification of linear temporal logic specifications. *Proc. 25th Int. Colloq. Aut. Lang. Prog.*, LNCS 1443, pages 1–16, 1998.

12. Y. Kesten, A. Pnueli, L. Raviv, and E. Shahar. LTL Model Checking with Strong Fairness. Technical Report mcs01-07, The Weizmann Institute of Science, 2001. Submitted to *Formal Methods in System Design*.
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.*, LNCS 115, pages 264–277, 1981.
14. O. Lichtenstein and A. Pnueli. Checking that finite-state concurrent programs satisfy their linear specification. *POPL'85*, pages 97–107.
15. Z. Manna, A. Anuchitanukul, N. Bjørner, A. Browne, E. Chang, M. Colón, L. D. Alfaro, H. Devarajan, H. Sipma, and T. Uribe. STeP: The Stanford Temporal Prover. Stanford University, 1994.
16. Z. Manna and A. Pnueli. Completing the temporal picture. *Theor. Comp. Sci.*, 83(1):97–130, 1991.
17. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer Verlag, New York, 1991.
18. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.
19. K. Namjoshi. Certifying model checkers. *CAV'01*, LNCS 2102.
20. D. Peled, A. Pnueli, and L. Zuck. From falsification to verification. *FTTCS'01*, LNCS 2245, pages 292–304.
21. A. Pnueli and R. Rosner. A framework for the synthesis of reactive modules. *Concurrency 88*, LNCS 335, pages 4–17.
22. M. Reynolds. An axiomatization of full computation tree logic. *Journal of Symbolic 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. F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. *CAV'00*, LNCS 1855, pages 248–263.
25. C. Sprenger. *On the Verification of CTL* Properties of Infinite-State Reactive Systems*. PhD thesis, Swiss Federal Institute of Technology, Lausanne, 2000.
26. 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.
27. M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Inf. and Comp.*, 115(1):1–37, 1994.