Verification of Temporal Properties

Limor Fix
Orna Grumberg

TR 93-1368
August 1993

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

Verification of temporal properties

Limor Fix

Upson Hall, Computer Science Dept.
Cornell University, Ithaca, NY 14850
e-mail:fix@cs.cornell.edu
Tel:(607)(255-9223)
Fax:(607)(255-4428)

Orna Grumberg

AT&T Bell Laboratories
Murray Hill, NJ 07974
e-mail:orna@research.att.com
Tel:(908)(582-5641)
Fax:(908)(582-7550)

July 20, 1993

Abstract

The paper presents a relatively complete deductive system for proving branching time temporal properties of reactive programs. No deductive system for verifying branching time temporal properties has been presented before. Our deductive system enjoys the following advantages. First, given a well-formed specification there is no need to translate it into a normal-form specification since the system can handle any well-formed specification. Second, given a specification to be verified, the proof rule to be applied is easily determined according to the top level operator of the specification. Third, the system reduces temporal verification to assertional reasoning rather than to temporal reasoning.

1 Introduction

Temporal logics are widely accepted and frequently used for specifying concurrent and reactive programs. In recent years, many fully automatic methods for verifying temporal specifications have been presented such as model checkers [4]. However, the scope of these methods is still very limited; the fully automatic methods mainly apply to finite state programs and to special cases of infinite state programs. Therefore, the main tool for establishing that a program satisfies its temporal specification is still that of deductive verification, using a set of axioms and inference rules.

Deductive verification can also be aided by the computer. A deductive verification system can easily be embedded in automated theorem provers, like Nuprl [5], Hol [8], Boyer-Moore [3] and Coq [6]. An automated theorem prover is an interactive environment for proof generation. It assists the development of proofs by exploring the possible proof steps, checking and writing intermediate results and assembling the solution.

We present a relatively complete deductive system for verifying fair branching-time temporal logic specifications (fair CTL). No deductive verification system has been presented before for fair CTL. All previous deductive systems for verifying temporal properties, e.g., [16],[9],[17],[12],[15], are concerned only with linear temporal logic (LTL). The previous deductive systems also suffer from the following drawback. They offer a relatively complete deductive system only for normal-form formulas. Thus, all other properties whose expression in LTL does not fall into the restricted normal-form can be verified only by translating them into normal-form formulas. The known method for translating an arbitrary (future) LTL formula into a normal-form is very complex in both the time complexity of the translation and the size of the resulting formula. First a tableau method is used to translate a future formula into a counter-free w-automata and then this automaton is translated into a normal-form formula [11], [18]. In contrast, our deductive system can handle an arbitrary nesting of temporal operators in a formula while no normal-form is required.

Our deductive system also enjoys the following two advantages. First, given a specification to be verified the possible rules or axioms to be applied are solely determined by the top level operator of the specification. Moreover, in most cases, the next possible rule to be applied is uniquely defined. This property of the deduction system is very helpful when embedding the system in an automated theorem prover. Second, all rules in our system reduce the task of verifying a temporal property into subgoals that either require proving the validity of assertional formulas or the verification of simpler temporal properties. In other words, none of the generated subgoals require proving validity of temporal formulas.

Next we describe our work in some more details. The deduction system proves validity of *correctness* formulas of the form "$P\ Sat\ p \rightarrow f$", where P is a program, p is a precondition given in some assertional language and f is a fair CTL formula. A program is defined as a set of transitions. A program step is executed by choosing nondeterministically, in a *weakly fair* manner [7], an enabled transition for execution. The weak fairness guarantees that, every constantly enabled transition is eventually chosen for execution. Formulas of fair CTL are interpreted over a node in a computation tree of a program. Every temporal operator consists of a path quantifier together with one modal operator. A path quantifier is either A for "all fair paths" or E for "there exists a fair path". A modal operator is either X for "next-state", G for "globally" or U, for "until". A correctness formula "$P\ Sat\ p \rightarrow f$" is valid iff for every computation tree of P, the root node satisfies $p \rightarrow f$, where $\rightarrow$ denotes implication (defined as usual).

Of special interest is the rule for verifying the formula $P\ Sat\ p \rightarrow EGf_1$. This formula specifies the existence of a fair infinite path in the computation tree of P along which f_1 is continuously satisfied.

We prove that an infinite path is fair by showing that it consists of infinitely many finite *fair* intervals. A fair interval is an interval along which every transition is either disabled or executed. To establish that, we introduce a proof tool for identifying the end points of fair intervals and in addition we formulate an inductive argument that implies infinitely many occurrences of such end points along the path (see page 6).

The rest of the paper is organized as follows. In Section 2 the computation model is presented. Section 3 defines fair *CTL* and correctness formulas. Section 4 presents the deduction system and an example is given in Section 5. In Section 6 we compare our deduction system with *CTL* model checking and discuss other verification approach.

2 The computation model

The model of computation we consider is a *fair transition system* in which each transition τ is a binary relation over a set of states Σ. $\sigma_1 \tau \sigma_2$ is used to denote that $(\sigma_1, \sigma_2) \in \tau$. We say that a transition τ is *enabled* in a state σ if there exists a state σ' such that $\sigma \tau \sigma'$. Otherwise, τ is *disabled*. We denote by $En(\tau)$ the set of all states in which τ is enabled. A *program* P over a set of states Σ is a set of transitions over Σ. We assume the existence of a *dummy* transition, τ^*, which is enabled exactly when all other transitions are disabled and which leaves the program state unchanged. The dummy transition ensures that all computations of the program are infinite.

Next we formally define the meaning of a program as a set of marked trees. A *node* η is a finite sequence over the natural numbers. A *tree* T is a set of nodes closed under the prefix operation. A node η is an *immediate successor* of a node ξ if there exists a natural number n such that $\xi\ n = \eta$. The *root* of a tree is the empty sequence. An *edge* e is a pair of nodes (ξ, η) such that η is an immediate successor of ξ. A *path* π from a node η is an infinite sequence $\eta_1 \eta_2 \ldots$ such that $\eta_1 = \eta$ and for all $i \geq 0$, η_{i+1} is an immediate successor of η_i. A *marked tree* is a triple $< T, M_n, M_e >$, where T is a tree, M_n is a function that maps every node of T to a state in Σ. If $M_n(\eta) = s$ then we say that η is *marked* by s. M_e is a function that maps every edge in T to a transition of P. A marked tree $< T, M_n, M_e >$ is a *computation tree* of P iff the set of immediate successors of every node η_1 in T is marked exactly by the set of all states that are reachable from $M_n(\eta_1)$ via the execution of a single transition of P. More formally, for every node η_1 in T and for every state s in Σ and for every transition τ of P:

$$(M_n(\eta_1), s) \in \tau$$
$$\textit{iff}$$
$$\exists \eta_2 \textit{ an immediate successor of } \eta_1 : M_n(\eta_2) = s \wedge M_e(\eta_1, \eta_2) = \tau$$

Finally, the *meaning* of a program P is the set of all computation trees of P.

A transition τ is *enabled* in a node η in a computation tree $< T, M_n, M_e >$ iff τ is enabled in $M_n(\eta)$. τ is *executed* along a path $\pi = \eta_1 \eta_2 \ldots$ of the computation tree iff there exists $i > 0$ such that $M_e(\eta_i, \eta_{i+1}) = \tau$. A path $\pi = \eta_1, \eta_2 \ldots$ in a computation tree of P is *fair* iff for every transition τ of P, if τ is continuously enabled from some point along π then τ is infinitely often executed along π. Note that, every finite prefix of a path can be extended to a fair path and that every path with an infinite suffix of τ^* executions is fair.

3 Fair CTL and correctness formulas

Assume an assertional language L whose formulas are interpreted over Σ. [1] A fair CTL formula is either a formula from L or, $\neg f_1$, $f_1 \wedge f_2$, AXf_1, EXf_1, AGf_1, EGf_1, $A[f_1 \mathcal{U} f_2]$, and $E[f_1 \mathcal{U} f_2]$, where f_1 and f_2 are fair CTL formulas. Fair CTL formulas are interpreted over a node in a marked tree. Given a node η, a marked tree MT and a fair CTL formula f, the satisfaction relation $MT, \eta \models f$ is defined by induction on the structure of the formula. Intuitively, an assertion p in L is satisfied at a node η iff the state that marks η, that is $M_n(\eta)$, satisfies p. $\neg f$ and $f_1 \wedge f_2$ are defined as usual. AXf_1 (EXf_1) is satisfied at η iff every (at least one) immediate successor of η satisfies f_1. AGf_1 (EGf_1) is satisfied at η iff every node in every (at least one) fair path from η satisfies f_1. Finally, $A[f_1 \mathcal{U} f_2]$ ($E[f_1 \mathcal{U} f_2]$) is satisfied at η iff every (at least one) fair path from η satisfy f_1 until f_2, i.e., there exists a node η' along the path that satisfies f_2 and every node from η to η' satisfies f_1. The set of operators presented above is not minimal, for example, the operators AG and EG can be expresses in terms of $A\mathcal{U}$ and $E\mathcal{U}$, respectively. We choose to introduce a wider set than necessary in order to simplify the presentation of the proof rules.

A *fair CTL correctness* formula consists of three components: a precondition p in L, a program P and a fair CTL formula f, and is of the form "P *Sat* $p \rightarrow f$". A formula "P *Sat* $p \rightarrow f$" is interpreted over the root node of a computation tree of P. A fair CTL correctness formula is *valid*, to be denoted $\models P$ *Sat* $p \rightarrow f$, iff for every computation tree of P the root node satisfies $p \rightarrow f$.

An *assertional correctness* formula consists also of three components: p and q in L and a set of transitions Γ, and is of the form "$\{p\}\Gamma\{q\}$". A formula "$\{p\}\Gamma\{q\}$" is interpreted over a pair of states (σ_1, σ_2) such that there exists $\tau \in \Gamma$ for which $\sigma_1 \tau \sigma_2$ holds. An assertional correctness formula is *valid*, to be denoted $\models \{p\}\Gamma\{q\}$, iff for every transition τ in Γ and every pair of states (σ_1, σ_2) such that $\sigma_1 \tau \sigma_2$ holds: *if* $\sigma_1 \models p$ *then* $\sigma_2 \models q$.

4 The deduction system

In this section we present our deductive system. Proof rules of special interest are explained in details and their soundness is motivated. The completeness proof is postponed to Appendix A.

4.1 The *next*-rules

To verify a specification of the form P *Sat* $p \rightarrow AXf_1$, we require that every transition of the program P that starts in a state satisfying p results in a state satisfying an assertion q. And moreover, if P' denotes the program left to be executed after the execution of a single step of P then every root node of a computation tree of P' that satisfies q should also satisfy f_1. Since a program in our model has a single control point the program left to be executed after performing a single step of the program, is the program itself. Therefore, we get:

$$\frac{\{p\}P\{q\} \qquad P \text{ Sat } q \rightarrow f_1}{P \text{ Sat } p \rightarrow AXf_1}$$

[1] We assume L is expressible enough to formalize all the sets of states required for the relative completeness of our system. As is known [19],[14], L should at least include the predicate calculus, interpreted symbols for expressing the standard operations and relations over integers and the fixed-point operators μ and ν.

To verify a specification of the form $P\ Sat\ p \rightarrow EXf_1$, we require that there exists a transition τ in P such that τ is enabled in all states satisfying p and its execution results in a state satisfying an assertion q. And moreover, every root node of a computation tree of P that satisfies q also satisfies f_1:

$$\frac{\begin{array}{l}\textit{There exists } \tau : p \rightarrow En(\tau) \textit{ and } \{p\}\tau\{q\} \\ P\ Sat\ q \rightarrow f_1\end{array}}{P\ Sat\ EXf_1}$$

To verify the negations of the above two specifications we relay on the following fair CTL validities:

$$\begin{array}{l}\neg AXf_1 \leftrightarrow EX\neg f_1 \\ \neg EXf_1 \leftrightarrow AX\neg f_1\end{array}$$

Thus we get:

$$\frac{P\ Sat\ p \rightarrow EX\neg f_1}{P\ Sat\ p \rightarrow \neg AXf_1}$$

and

$$\frac{P\ Sat\ p \rightarrow AX\neg f_1}{P\ Sat\ p \rightarrow \neg EXf_1}$$

4.2 The *until*-rules

Next we present conditions for verifying $P\ Sat\ p \rightarrow A[I_1 \mathcal{U} I_2]$, where I_1 and I_2 are in L. Let a prefix of a path in which all nodes satisfy $\neg I_2$ be called *I_2-avoiding.* We have to verify that all I_2-avoiding prefixes that start in roots satisfying p are finite and that I_1 is continuously satisfied along these prefixes. The following verification conditions establish a well-founded induction on the length of the I_2-avoiding prefixes. The induction hypothesis assumes that all nodes along I_2-avoiding prefixes satisfy some state predicate Φ and that a ranking function δ is defined for all states that mark these nodes, where δ maps states into a well-founded, partially ordered set $(\mathcal{W}, \leq)$. Moreover, the induction hypothesis assumes that the ranks defined by δ along an I_2-avoiding prefix never increase. In the induction basis we deal with the case of I_2-avoiding prefixes of length zero. We require that every state that satisfies p also satisfies either I_2 or it satisfies Φ and δ is defined for that state (denoted by $\delta \in \mathcal{W}$):

$$AU1.\ \ p \rightarrow (I_2 \vee (\Phi \wedge (\delta \in \mathcal{W})))$$

In the induction step we require that every transition of the program that starts in a state satisfying Φ and for which a rank w is defined by δ results in a state that either satisfies I_2 or it satisfies Φ and it is mapped by δ to a rank lower or equal to w:

$$AU2.\ \ \{\Phi \wedge (\delta = w)\}P\{I_2 \vee (\Phi \wedge (\delta \leq w))\}$$

We add the requirement that every state that satisfies Φ also satisfies I_1:

$$AU3.\ \ \Phi \rightarrow I_1$$

Conditions $AU1$-$AU3$ guarantee that every path in a computation tree of P from a root satisfying p satisfies I_1 as long as I_2 is not satisfies. To ensure that I_2 will eventually be satisfied we relay on the fairness of the computation model, the well-foundedness of $\mathcal{W}$ and the additional requirement that for every state that satisfies $\Phi \wedge (\delta = w)$ there exists an enabled transition of the program whose execution results in a state that either satisfies I_2 or satisfies Φ and for which a lower rank than w is defined by δ:

$$\begin{array}{ll} AU4. & \textit{For every } w \in \mathcal{W} \textit{ there exists } \tau \in P: \\ & (1)\ (\Phi \wedge (\delta = w)) \rightarrow En(\tau) \\ & (2)\ \{\Phi \wedge (\delta = w)\}\tau\{I_2 \vee (\Phi \wedge (\delta < w))\} \end{array}$$

The fairness of the computation model implies that a transition that causes the rank to decrease will eventually be executed and the well-foundness of $\mathcal{W}$ guarantees that only finitely many times the rank can decrease and therefore a node satisfying I_2 must be reached. Thus, we get:

$$\frac{AU1 - AU4}{\begin{array}{l} P\ \ Sat\ \ p \rightarrow A[I_1 \mathcal{U} I_2] \\ \textit{where } I_1, I_2 \in L \end{array}}$$

A specification $P\ Sat\ p \rightarrow E[I_1 \mathcal{U} I_2]$, where $I_1, I_2 \in L$, is verified by the above verification conditions $AU1$, $AU3$ and $AU4$. Condition $AU2$ is omitted in order to relax the set of verification conditions such that they only imply the existence of a fair path with a finite prefix $\eta_0\eta_1 \ldots \eta_i$ such that $\delta(M_n(\eta_0)) > \delta(M_n(\eta_1)) > \ldots > \delta(M_n(\eta_i))$, η_i satisfies I_2 and every other node along this prefix satisfies I_1:

$$\frac{AU1, AU3, AU4}{\begin{array}{l} P\ \ Sat\ \ p \rightarrow E[I_1 \mathcal{U} I_2] \\ \textit{where } I_1, I_2 \in L \end{array}}$$

To verify a specification of the form $A[f_1 \mathcal{U} f_2]$, where either f_1 or f_2 is not in L, we decompose the verification task into three subtasks. One requires that every path in a computation tree of P that starts in a node satisfying p satisfies I_1 until I_2, where both I_1 and I_2 are in L. The other two require that every root of a computation tree of P that satisfies I_1 or I_2 also satisfies f_1 or f_2, respectively:

$$\frac{\begin{array}{l} P\ Sat\ p \rightarrow A[I_1 \mathcal{U} I_2] \\ P\ Sat\ I_1 \rightarrow f_1 \\ P\ Sat\ I_2 \rightarrow f_2 \end{array}}{\begin{array}{l} P\ Sat\ p \rightarrow A[f_1 \mathcal{U} f_2] \\ \textit{where } f_1 \notin L \textit{ or } f_2 \notin L. \end{array}}$$

Again the soundness of this rule relies on the fact that a program has a single control point and the program left to be executed after performing one or more steps, is the program itself.

To verify a specification of the form $P\ Sat\ p \rightarrow \neg A[f_1 \mathcal{U} f_2]$ we relay on the following fair CTL valid formula:

$$\neg A[f_1 \mathcal{U} f_2] \leftrightarrow (EG\neg f_2 \vee E[(\neg f_2)\mathcal{U}(\neg f_1 \wedge \neg f_2)])$$

Thus we get:

$$\frac{\begin{array}{l} P\ Sat\ p \rightarrow EG\neg f_2 \\ or \\ P\ Sat\ p \rightarrow E[(\neg f_2)\mathcal{U}(\neg f_1 \wedge \neg f_2)] \end{array}}{P\ Sat\ \neg A[f_1 \mathcal{U} f_2]}$$

To verify a specification of the form $P\ Sat\ p \rightarrow \neg E[f_1 \mathcal{U} f_2]$ we observe that there does not exist a fair path that satisfies f_1 until f_2 iff either $\neg f_1 \wedge \neg f_2$ holds at the root or there exists an assertion I that holds in every node of every path from the root until $\neg f_1 \wedge \neg f_2$ holds and in addition I must imply $\neg f_2$. Thus we get:

$$\frac{\begin{array}{l} P\ Sat\ p \rightarrow (\neg f_1 \wedge \neg f_2) \\ or \\ p \rightarrow I \\ P\ Sat\ I \rightarrow \neg f_2 \wedge AX(I \vee (\neg f_1 \wedge f_2)) \end{array}}{P\ Sat\ p \rightarrow \neg E[f_1 \mathcal{U} f_2]}$$

4.3 The *global*-rules

Next we present conditions for verifying $P\ Sat\ p \rightarrow EGf_1$. To prove that a path is fair we exploit the following observation, taken from the completeness proof for the weak fair termination rule in [10]: a path π in a computation tree of P is fair iff for every transition τ_i of P, either τ_i is infinitely often disabled or τ_i is infinitely often executed along π. Note that, along every path the dummy transition τ^* is either disabled or continuously executed from some point on. Thus, we can relax the above condition and conclude that a path π is fair iff for every non-dummy transition τ_i, i.e., τ_i is not equal to τ^*, either τ_i is infinitely often disabled or τ_i is infinitely often executed along π. This implies that π can be partitioned into infinitely many disjoint intervals of finite length, each of which contains for every non-dummy transition τ_i, either a state in which τ_i is disabled or a step in which τ_i is executed. We call such an interval *fair*. Thus, a path is fair iff it can be partitioned into infinitely many finite fair intervals.

A proof tool for identifying the end points of fair intervals, is introduced next. Let P be a program with m non-dummy transitions $\tau_1, \ldots, \tau_m$ and let $dis : \Sigma \rightarrow \{0,1\}^m$ be a function that maps a state σ to a binary vector of length m such that $dis(\sigma)(j) = 0$ iff the transition τ_j is disabled in σ. Let $\overline{0}$ ($\overline{1}$) stands for a vector of m zeros (ones). And for a natural number j let $\widehat{j}$ be a vector of m ones except that if $1 \leq j \leq m$ then the j-th element in this vector is zero. Let $\triangle$ be the point wise logical conjunction of binary vectors. For example, let $m = 3$ and $dis = 101$, the value of the expression $\overline{1} \triangle \widehat{3} \triangle dis$, that is $111 \triangle 110 \triangle 101$, is equal to 100. We use a function g from the program states to $\{0,1\}^m$ and require the following proof obligations that ensure that $g = \overline{0}$ indicates the end of a fair interval. The condition

$$EG1.\ p \rightarrow (g \in \{0,1\}^m)$$

requires that initially g is defined. The condition

$$EG2.\ \ For\ every\ \tau_j \in P:\ \{g = \overline{0}\}\tau_j\{g = (\widehat{j} \triangle dis)\}$$

requires that the first step taken after the end of a fair interval results in a state in which the value of g is reset, that is, $g = (w_1, \ldots, w_m)$ where $w_i = 0$ iff τ_i is disabled at the current state or τ_i has just been executed and $w_i = 1$, otherwise. The condition

$$EG3.\ \textit{For every } \tau_j \in P \textit{ and every } \overline{w} \in \{0,1\}^m: \quad \{g = \overline{w} \wedge \overline{w} \neq \overline{0}\}\tau_j\{g = (\overline{w} \triangle \hat{j} \triangle dis)\}$$

requires that g assigns $w_i = 0$ to states within a fair interval iff τ_i has either been executed or has been disabled in that fair interval.

Introducing the above method for identifying the end points of fair intervals, we still have to prove that there exists a path along which infinitely often an end of a fair interval is encountered:

$$\begin{array}{ll} EG4. & I \rightarrow g = \overline{0} \\ EG5. & P\ Sat\ p \rightarrow E[f_1 \mathcal{U} (I \wedge f_1)] \\ EG6. & P\ Sat\ I \rightarrow EX\,E[f_1 \mathcal{U} (I \wedge f_1)] \end{array}$$

The condition $EG4$ sets the connection between the satisfaction of I and the end points of fair intervals. Conditions $EG5$ and $EG6$ ensure that there exists a path in which I holds infinitely often and moreover f_1 continuously holds along that path. Thus we get:

$$\frac{EG1 - EG6}{P\ \ Sat\ \ p \rightarrow EG f_1}$$

To verify the other global specifications we relay on the following fair CTL validities,

$$\begin{array}{rcl} AG f_1 & \leftrightarrow & \neg E[true\ \mathcal{U} \neg f_1] \\ \neg EG f_1 & \leftrightarrow & A[true\ \mathcal{U} \neg f_1] \\ \neg AG f_1 & \leftrightarrow & E[true\ \mathcal{U} \neg f_1] \end{array}$$

which imply:

$$\frac{P\ Sat\ p \rightarrow \neg E[true\ \mathcal{U} \neg f_1]}{P\ Sat\ p \rightarrow AG f_1} \qquad \frac{P\ Sat\ p \rightarrow A[true\ \mathcal{U} \neg f_1]}{P\ Sat\ p \rightarrow \neg EG f_1} \qquad \frac{P\ Sat\ p \rightarrow E[true\ \mathcal{U} \neg f_1]}{P\ Sat\ p \rightarrow \neg AG f_1}$$

The entire deductive system is presented in Figure 1.

5 Example

Consider the simple program,

$$\begin{array}{llll} P :: & \tau_1 : & x := x + 1 & \textit{if} \quad x < 10 \\ & [\!] & & \\ & \tau_2 : & y := 5 & \textit{if} \quad 5 < x \wedge y = 0 \\ & [\!] & & \\ & \tau_3^* & & \end{array}$$

which has three transitions. Transition τ_1 increases the value of x by 1 and is enabled whenever the value of x is smaller than 10. Transition τ_2 sets y to 5 and is enabled whenever the value of x is bigger than 5 and the value y is equal to 0. Transition τ_3^* is the dummy transition. Next we verify the correctness formula:

$$P \ Sat \ x = 0 \wedge y = 0 \rightarrow EG(x < 10 \rightarrow y = 0)$$

This specification implies that there exists a fair computation of P in which the execution of the second transition is postponed until the first one is not enabled any more. Let

$$g = \begin{cases} 00 & \textit{if } 0 \leq x \leq 5 \vee y \neq 0 \\ 01 & \textit{if } x > 5 \wedge y = 0 \end{cases}$$

We prove that the premises $EG1 - EG6$ hold:

- $EG1$. According to the definition of g, $x = 0 \wedge y = 0 \rightarrow g = 00$ holds.
- $EG2$. According to the definition of g if $g = 00$ then $0 \leq x \leq 5 \vee y \neq 0$ holds. Therefore, transition τ_1 is either not enabled or its execution results in a state in which $g = 0c$, where $c = 1$ if $x > 5 \wedge y = 0$ and $c = 0$ otherwise. According to the definition of dis, the value of the expression $01 \bigtriangleup dis$ is $0d$, where $d = 1$ if $x > 5 \wedge y = 0$ and $d = 0$ otherwise. Thus, in the resulting state $g = (01 \bigtriangleup dis)$ and we conclude that

  $$\{g = 00\}\tau_1\{g = (01 \bigtriangleup dis)\}$$

  holds. Transition τ_2 is not enabled in a state satisfying $0 \leq x \leq 5 \vee y \neq 0$. Therefore

  $$\{g = 00\}\tau_2\{g = (10 \bigtriangleup dis)\}$$

  holds. For transition τ_3^*

  $$\{g = 00\}\tau_3^*\{g = (11 \bigtriangleup dis)\}$$

  holds since either τ_3^* is not enabled or it is enabled and in both starting and resulting states $dis = 00$ and $g = 00$.
- $EG3$. According to the definition of g, $g = \overline{w} \wedge \overline{w} \neq \overline{0}$ implies that $g = 01$ and thus the corresponding starting state satisfies $x > 5 \wedge y = 0$. Transition τ_1 is either not enabled or its execution results in a state satisfies $x > 5 \wedge y = 0$. Therefore according to the definition of g in the resulting state $g = 01$ holds. The value of the expression $01 \bigtriangleup 01 \bigtriangleup dis$ is equal to 01 since τ_2 is enabled in the resulting state and thus

  $$\{g = 01\}\tau_1\{g = (01 \bigtriangleup 01 \bigtriangleup dis)\}$$

  holds. The execution of τ_2 from a state satisfying $x > 5 \wedge y = 0$ results in a state satisfying $x > 5 \wedge y \neq 0$ and therefore $g = 00$ in the resulting state. The value of the expression $01 \bigtriangleup 10 \bigtriangleup dis$ is equal to 00 and thus

  $$\{g = 01\}\tau_2\{g = (01 \bigtriangleup 10 \bigtriangleup dis)\}$$

  holds. The transition τ_3^* is not enabled in a state satisfying $x > 5 \wedge y = 0$ and therefore

  $$\{g = 01\}\tau_3^*\{g = (01 \bigtriangleup 11 \bigtriangleup dis)\}$$

  holds.

- $EG4$. Let $I = (0 \leq x \leq 5 \land y = 0) \lor (x \geq 10 \land y \neq 0)$. According to the definition of g we get:

$$I \rightarrow g = 00$$

- $EG5$. Next we prove that

$$P\ Sat\ x = 0 \land y = 0 \rightarrow E[(x < 10 \rightarrow y = 0)\mathcal{U}(I \land (x < 10 \rightarrow y = 0))]$$

Using first order manipulation the assertion $(I \land (x < 10 \rightarrow y = 0))$ can we rewritten as $(0 \leq x \leq 5 \land y = 0) \lor (y \neq 0 \land x \geq 10)$. Let $\Phi = 0 \leq x \leq 10 \land y = 0$ and $\mathcal{W} = \{true, false\} \times \{0..10\}$ with lexicographical order where $< false, 0 >$ is the minimal element. We define $\delta =< y = 0, 10-x >$.

  - $AU1$. $x = 0 \land y = 0 \rightarrow \Phi \land (\delta \in \mathcal{W})$.
  - $AU3$. $0 \leq x \leq 10 \land y = 0 \rightarrow (x < 10 \rightarrow y = 0)$.
  - $AU4$. For $\delta =< true, d >$, where $1 \leq d \leq 10$ we get

$$(1)\ \Phi \land \delta =< true, d > \land 1 \leq d \leq 10 \rightarrow En(\tau_1)$$
$$(2)\ \{\Phi \land \delta =< true, d > \land 1 \leq d \leq 10\}\tau_1\{\Phi \land \delta =< true, d-1 >\}$$

For $\delta =< true, 0 >$, we get

$$(1)\ \Phi \land \delta =< true, 0 > \rightarrow En(\tau_2)$$
$$(2)\ \{\Phi \land \delta =< true, 0 >\}\tau_2\{y \neq 0 \land x \geq 10 \land \delta =< false, 0 >\}$$

For $\delta =< false, d >$, the assertion $\Phi \land \delta =< false, d >$ is *false* and therefore both requirements (1) and (2) hold for any transition.

- $EG6$. Next we prove that

$$(*)\ P\ Sat\ I \rightarrow EXE[(x < 10 \rightarrow y = 0)\mathcal{U}(I \land (x < 10 \rightarrow y = 0))]$$

Recall that $I = (0 \leq x \leq 5 \land y = 0) \lor (x \geq 10 \land y \neq 0)$ therefore (*) holds iff

$$(1)\ \ P\ Sat\ 0 \leq x \leq 5 \land y = 0 \rightarrow EXE[(x < 10 \rightarrow y = 0)\mathcal{U}(I \land (x < 10 \rightarrow y = 0))]$$

holds and

$$(2)\ \ P\ Sat\ x \geq 10 \land y \neq 0 \rightarrow EXE[(x < 10 \rightarrow y = 0)\mathcal{U}(I \land (x < 10 \rightarrow y = 0))]$$

holds.

To prove (1) we apply the *next*-rule for proving EX and get the following subgoals:

$$(1.1)\ \ 0 \leq x \leq 5 \land y = 0 \rightarrow x < 10$$
$$(1.2)\ \ \{0 \leq x \leq 5 \land y = 0\}\tau_1\{0 \leq x \leq 6 \land y = 0\}$$
$$(1.3)\ \ P\ Sat\ 0 \leq x \leq 6 \land y = 0 \rightarrow E[(x < 10 \rightarrow y = 0)\mathcal{U}(I \land (x < 10 \rightarrow y = 0))]$$

It is easy to see that (1.1) and (1.2) hold and (1.3) is proved using Φ and δ just as in $EG5$.

To prove (2) we apply the rule for proving EX and get the following subgoals:

$$(2.1)\ \ x \geq 10 \land y \neq 0 \Rightarrow En(\tau_3^*),\ where\ En(\tau_3^*) = x \geq 10 \land y \neq 0$$
$$(2.2)\ \ \{x \geq 10 \land y \neq 0\}\tau_3^*\{x \geq 10 \land y \neq 0\}$$
$$(2.3)\ \ P\ Sat\ x \geq 10 \land y \neq 0 \rightarrow E[(x < 10 \rightarrow y = 0)\mathcal{U}(I \land (x < 10 \rightarrow y = 0))]$$

It is easy to see that (2.1) and (2.2) hold and (2.3) is easily proved using $\Phi = false$ and any δ. We can choose Φ and δ as such since the precondition implies the second argument of the until specification, that is, $x \geq 10 \land y \neq 0 \rightarrow I \land (x < 10 \rightarrow y = 0)$.

6 Discussion

CTL model checking [4] is a verification algorithm that given a program described as a finite-state transition graph, a state in the program, and a formula in propositional *CTL*, determines whether or not the computation tree of the program, starting at this state, satisfies the formula. It is interesting to notice the similarity and difference between the model checking approach and ours.

Both methods are similar in that, the verification of a formula depends on the verification of its subformulas. Moreover, they are both syntax directed, i.e., the rule (or procedure) applied in the verification of a formula is determined by the formula's top level operator.

One of the differences between the methods stems from the fact that while [4] solves a recursive problem, we suggests a method for a non-recursive one. As a result, model checking suggests no special rules for negated formulas. To determine whether or not a formula $\neg\Phi$ is true in a state they check Φ at this state and complement the result. Dealing with a non-recursive problem, our method cannot expect, in general, to get a negative answer. Thus direct rules to handle negation as the top level operator are introduced.

In [4], a more general notion of fairness is considered. Therefore, handling fairness requires a preliminary step, that marks all state from which a fair computation starts (they all satisfy some proposition Q). To check now that $E[f_1 \mathcal{U} f_2]$ is true in a state, they check that $E[f_1 \mathcal{U}(f_2 \wedge Q)]$ is true in that state. In this case, our method is simpler. Since from every state there is a fair weak computation starting at this state, we can verify $E[f_1 \mathcal{U} f_2]$ as if no fairness is concerned.

The case of $A\mathcal{U}$ is solved in [4] by using $E\mathcal{U}$ and EG. Their procedure for EG heavily depends on the finiteness of the program description, and involves graph manipulations. Clearly, a similar method is not applicable to our case. To conclude, both methods are similar in the way they take advantage of the structure of *CTL* formulas. As expected, they diverge significantly in the way they exploit properties of the program description.

Other verification approach that can handle general liveness properties were introduced in the automata theoretic framework. In [1],[2],[13] assertional verification conditions are presented for verifying properties which are specified by finite-state automata. Those results are extended in [20] to deal with properties specified by recursive ω-automata. In contrast, we specify properties in a relatively intuitive and high level temporal language and no automata is constructed.

Acknowledgments.
We thank Nissim Francez for fruitful discussions in the preliminary stage of this work.

References

[1] B. Alpern and F.B. Schneider. Proving boolean combinations of deterministic properties. In *2nd IEEE Symp. on Logic in Computer Science*, pages 131–137, 1987.

[2] B. Alpern and F.B. Schneider. Verifying temporal properties without temporal logic. *ACM Trans. Prog. Lang. Sys.*, 11:147–167, 1989.

[3] R.S. Boyer and J.S. Moore. *Integrating Recursion Procedures into Heuristic Theorem Provers: A Case Study in Linear Arithmetic.* Oxford University Press, 1988.

[4] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Systems*, 8(2):244 – 263, 1986.

[5] R. L. Constable et al. *Implementing Mathematics with the Nuprl Development System.* Prentice-Hall, NJ, 1986.

[6] T. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76:95–120, 1988.

[7] N. Francez. *Fairness.* Springer-Verlag, 1986.

[8] M. Gordon. Hol: A machine oriented formalization of higher order logic. Technical Report 68, Cambridge University, 1985.

[9] L. Lamport. The hoare logic of concurrent programs. *Acta Informatica*, 14, 1980.

[10] D. Lehmann, A. Pnueli, and J. Stavi. Impartiality, justice and fairness: the ethics of concurrent termination. In *8th ICALP.* LNCS 115, Springer-Verlag, 1981.

[11] O. Maler and A. Pnueli. Tight bounds on the complexity of cascaded decomposition of automata. In *In Proc. 31th IEEE Symp. Found. of Comp. Sci.*, pages 672–682, 1990.

[12] Z. Manna and A. Pnueli. How to cook a temporal proof system for your pet language. In *10th ACM Symp. Princ. of Prog. Lang.*, pages 141–154, 1983.

[13] Z. Manna and A. Pnueli. Specification and verification of concurrent programs by ∀-automata. In *14th ACM Symp. on Principles of Programming Languages*, pages 1–12, 1987.

[14] Z. Manna and A. Pnueli. Completing the temporal picture. In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *16th Int. Colloq. Aut. Lang. Prog.* LNCS 372, Springer-Verlag, 1989. Also in Theor. Comp. Sci., 1991, 83(1):97-130.

[15] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems: specification.* Springer-Verlag, 1991.

[16] Z. Manna and R. Waldinger. Is sometime sometime better than always? intermittent assertions in proving program correctness. *Communications ACM*, 21(2):159–172, 1978.

[17] S. Owicki and L. Lamport. Proving liveness properties of concurrent programs. *ACM transactions on programming languages and systems*, 4(3):455–495, 1982.

[18] A. Pnueli and L. Zuck. In and out of temporal logic. In *In Proc. 8th IEEE Symp. Logic in Comp. Sci.*, pages 124–135, 1993.

[19] F.A. Stomp, W.-P. de Roever, and R.T. Gerth. The μ-calculus as an assertion language for fairness arguments. Technical Report 84-12, Utrecht, 1984.

[20] M. Vardi. Verification of concurrent programs: The automata-theoretic framework. *Annals of Pure and Applied Logic*, 51:79–98, 1991.

A Relative completeness

To prove the relative completeness of our deductive system we first prove that there is no circularity in the system. That is, we prove that every premise of every rule can be verified by applying less proof rules than required for the verification of the goal of the rule. To do so, we introduce a mapping ϱ that maps every fair CTL formula to a natural number. Then, we show that for every rule of the form

$$\frac{\begin{array}{l} P\ Sat\ p_1 \rightarrow \Phi_1 \\ \ldots \\ P\ Sat\ p_n \rightarrow \Phi_m \end{array}}{P\ Sat\ p \rightarrow \Phi}$$

the relation $\varrho(\Phi_i) < \varrho(\Phi)$ holds for every $1 \leq i \leq m$.

The function ϱ is:

- if $f \in L$ then $\varrho(f) = 1$,
- if $f = f_1 \wedge f_2$ then $\varrho(f) = \varrho(f_1) + \varrho(f_2)$,
- if $f = EXf_1$ or $f = AXf_1$ then $\varrho(f) = \varrho(f_1) + 1$,
- if $f = A[f_1 \mathcal{U} f_2]$ or $f = E[f_1 \mathcal{U} f_2]$ then $\varrho(f) = 2 \times (\varrho(f_1) + \varrho(f_2))$,
- if $f = EGf_1$ then $\varrho(f) = 4 \times \varrho(f_1) + 4$,
- if $f = AGf_1$ then $\varrho(f) = (2 \times (\varrho(f_1))^2 + 4)^2$,
- if $f = \neg f_1$ then $\varrho(f) = (\varrho(f_1))^2$.

Here we demonstrate the above for only one rule:

$$\frac{P\ Sat\ p \rightarrow A[true\ \mathcal{U} \neg f_1]}{P\ Sat\ \neg EGf_1}$$

According to the definition of ϱ we get:

$$\varrho(A[true\ \mathcal{U} \neg f_1]) = 2 \times (1 + (\varrho(f_1))^2)$$

and

$$\varrho(\neg EGf_1) = (4 \times \varrho(f_1) + 4)^2.$$

It is easy to see that $\varrho(\neg EGf_1) > \varrho(A[\ true\ \mathcal{U} \neg f_1])$.

Since there is no circularity in the system the relative completeness of the system can be proved by separately proving the relative completeness of every proof rule. We bring here some of the more interesting proofs.

- **The assertion rule:**

$$\frac{p \rightarrow q}{P\ Sat\ p \rightarrow q}$$

$$For\ q \in L$$

Assume $\models P\ Sat\ p \rightarrow q$. Thus for every computation tree of P if the root node satisfies p then it satisfies q as well. Since every state in the state space can serve as a root node for a computation tree of P we can conclude $\models p \rightarrow q$.

- **The AX rule:**

$$\frac{\begin{array}{l} P\ Sat\ q \rightarrow f_1 \\ \{p\}P\{q\} \end{array}}{P\ Sat\ p \rightarrow AXf_1}$$

Assume $\models P\ Sat\ p \rightarrow AXf_1$. Thus for every computation tree of P if the root node η satisfies p then every immediate successor of η satisfies f_1. Let q be an assertion that holds exactly at all immediate successors of root nodes that satisfy p. The program left to be executed after the execution of any transition from P is P itself, therefore, $\models P\ Sat\ q \rightarrow f_1$. Moreover, for every $\tau \in P$ and every pair of states (σ_1, σ_2) such that $\sigma_1 \tau \sigma_2$ holds: if $\sigma_1 \models p$ then $\sigma_2 \models q$, that is, $\models \{p\}P\{q\}$.

- **The $\neg AX$ rule:**

$$\frac{P\ Sat\ EX\neg f_1}{P\ Sat\ \neg AXf_1}$$

The relative completeness of this rule is a consequence of the validity of the fair CTL formula:

$$EX\neg f_1 \leftrightarrow \neg AXf_1$$

First direction, the formula $EX\neg f_1$ holds at a node η in a marked tree MT iff there exists an immediate successor η_1 of η such that η_1 does not satisfy f_1. Therefore not all immediate successors of η satisfy f_1, that is $MT, \eta \models \neg AXf_1$. The second direction is also easy, omitted here.

- **The $\neg A\mathcal{U}$ rule:**

$$\frac{\begin{array}{l} P\ Sat\ p \rightarrow EG\neg f_2 \\ or \\ P\ Sat\ p \rightarrow E[(\neg f_2)\mathcal{U}(\neg f_1 \wedge \neg f_2)] \end{array}}{P\ Sat\ \neg A[f_1 \mathcal{U} f_2]}$$

Again, the relative completeness of this rule is a consequence of the validity of the fair CTL formula:

$$(EG\neg f_2) \vee (E[(\neg f_2)\mathcal{U}(\neg f_1 \wedge \neg f_2)]) \leftrightarrow \neg A[f_1 \mathcal{U} f_2]$$

First direction, the formula $(EG\neg f_2) \vee (E[(\neg f_2)\mathcal{U}(\neg f_1 \wedge \neg f_2)])$ holds at a node η in a marked tree MT iff either there exists a fair path from η in which $\neg f_2$ continuously holds or there exists a fair path from η such that $\neg f_2$ holds in an initial prefix of that path until $\neg f_1 \wedge \neg f_2$ holds. Thus, we can conclude that there exists a fair path from η in which $f_1 \mathcal{U} f_2$ does not hold, that is, $MT, \eta \models \neg A[f_1 \mathcal{U} f_2]$. Using similar consideration the second direction can also be proved.

- **The EG rule:**

$$\frac{EG1 - EG6}{P\ Sat\ p \rightarrow EGf_1}$$

Assume $\models P\ Sat\ p \rightarrow EGf_1$. Thus for every computation tree of P if the root node satisfies p then there exists a fair path that continuously satisfy f_1. We translate P into another program P' by adding to P an history (auxiliary) variable h. Initially, the value of h is the empty sequence ϵ. Every transition τ_j in P is translated into

$$\tau_j \parallel h := h \circ \sigma \circ j$$

That is, the state at which τ_j is executed (i.e., σ) and the index number of τ_j (i.e., j) are concatenated to h.

Let $\mathcal{T}$ be the set of all computation trees of P' such that their root node is marked by a state satisfying $p \wedge h = \epsilon$.

For every state σ that marks a node in a tree in $\mathcal{T}$ we define the value of the function g as follows:

- The states that mark root nodes are mapped by g to the vector $\overline{1}$.
- If a node η is marked by σ and $g(\sigma) = \overline{w}$, where $\overline{w} \neq \overline{0}$, then every state σ' that marks an immediate successor η' of η is mapped by g to $\overline{w} \triangle \overline{j} \triangle dis$, where j is s.t. $M_e(\eta, \eta') = \tau_j$ and dis is evaluated at the state $M_n(\eta')$.
- If a node η is marked by σ and $g(\sigma) = \overline{0}$ then every state σ' that marks an immediate successor η' of η is mapped by g to $\widehat{j} \triangle dis$, where j is s.t. $M_e(\eta, \eta') = \tau_j$ and dis is evaluated at the state $M_n(\eta')$.

The above partial function g is well-defined since every two states that mark nodes in $\mathcal{T}$ (of either different trees or of the same tree) are different since h has different values.

We define I to be the set of all states that g maps to $\overline{0}$. Next we prove that the premises $EG1 - EG6$ hold for the above g and I and the precondition $p \wedge h = \epsilon$.

- $EG1$. Every state that satisfies $p \wedge h = \epsilon$ marks a root node of one of the trees in $\mathcal{T}$ and every such state is also mapped by g to $\overline{1}$, therefore

$$\models p \wedge h = \epsilon \rightarrow (g \in \{0,1\}^m)$$

- $EG2$. According to the definition of g:

$$\textit{For every } \tau_j \in P' : \{g = \overline{0}\}\tau_j\{g = (\widehat{j} \triangle dis)\}$$

- $EG3$. Again according to the definition of g:

$$\textit{For every } \tau_j \in P' \textit{ and every } \overline{w} \in \{0,1\}^m : \ \{g = \overline{w} \wedge \overline{w} \neq \overline{0}\}\tau_j\{g = (\overline{w} \triangle \widehat{j} \triangle dis)\}$$

- $EG4$. Immediate from the definition of I we get

$$\models I \rightarrow g = \overline{0}$$

- $EG5 - EG6$. Assuming the relative completeness of the other rules it is sufficient to prove that conditions $EG5 - EG6$ are semantically true. By the initial assumption we know that in every computation tree of P' that starts in a state satisfying $p \wedge h = \epsilon$ there exists a fair path that continuously satisfy f_1. According to the definition of I and g and the observation that every fair path can be divided into infinity many fair intervals we can conclude that I holds infinitely many times in that fair path. Therefore,

$$\models P\ Sat\ p \rightarrow E[f_1 \mathcal{U}(I \wedge f_1)]$$

and

$$\models P\ Sat\ I \rightarrow EXE[f_1 \mathcal{U}(I \wedge f_1)].$$

- **The basic-$\mathcal{AU}$ rule:**

$$\frac{AU1 - AU4}{\begin{array}{l} P\ Sat\ p \rightarrow A[I_1 \mathcal{U} I_2] \\ where\ I_1, I_2 \in L \end{array}}$$

Assume $\models P\ Sat\ p \rightarrow A[I_1 \mathcal{U} I_2]$. Given a computation tree $MT' =< T', M_n', M_e' >$ of P that starts in a root node η_0 satisfying p (i.e., $M_n'(\eta_0) \models p$) we know that every fair path from η_0 satisfies $I_1 \mathcal{U} I_2$. To prove the relative completeness of the rule we have to find an assertion Φ, a well-founded, partially ordered set $(\mathcal{W}, \leq)$ and a partial ranking function δ such that the premises $AU1. - AU4.$ hold.

We start by truncating MT' into a smaller tree $MT =< T, M_n, M_e >$ in the following way. Every fair path from the root is truncated exactly after the first node that satisfies I_2.

According to the assumption $\models P\ Sat\ p \rightarrow A[I_1 \mathcal{U} I_2]$ we know that MT has either infinite unfair paths or finite paths in which all intermediate nodes satisfy I_1 and the *leaf* nodes (nodes that have no successors) satisfy I_2. Next we construct another marked tree that will have only finite paths. First we need some definitions.

A path π is ***τ-avoiding*** if and only if τ is enabled at every node in π and moreover τ is not executed along π. A $CONE_\tau(\eta)$ is the set of all nodes in MT residing on infinite τ-avoiding paths starting from the node η. τ is called the CONE***'s directive.*** A path π in MT is *leaving* a $CONE_\tau(\eta)$ at a node η_1 if η_1 is in π and η_1 also belongs to $CONE_\tau(\eta)$ and the node which immediately follows η_1 in π does not belong to $CONE_\tau(\eta)$.

Next we inductively define the construction of another marked tree, to be denoted $MT^* =< T^*, M_n^*, M_e^* >$. The function M_n^* maps each node of T^* to a subset of T and the function M_e^* maps each edge of T^* to a transition of P.

At the base step we define the value of M_n^* for the root of the new tree MT^*. In the induction step we assume that the subtree of MT^* of depth n is already built. We define for each leaf ξ [2] of depth n the set of its immediate successors $\xi_1, \ldots, \xi_k$ in T^*. We also define for each successor ξ_j of ξ the value of $M_n^*(\xi_j)$ and the value of $M_e^*(\xi, \xi_j)$.

To define the base and the induction step we need a function $RT : T^* \rightarrow T$ which maps each node in T^* to a node in T. This function is also defined inductively. Let η_0 and ξ_0 denote the roots of MT and MT^*, respectively.

Base Step: If there exists in MT a τ-avoiding path starting from the root η_0 for some $\tau \in P$ then $M_n^*(\xi_0) = CONE_\tau(\eta_0)$. Else, $M_n^*(\xi_0) = \{\eta_0\}$. In both cases we define $RT(\xi_0) = \eta_0$.

[2] In the sequel nodes of MT are denoted by the symbols $\eta, \eta_0, \ldots$ or η' and the nodes of MT^* are denoted by the symbols $\xi, \xi_0, \ldots$ or ξ'.

Induction Step: Let ξ be a leaf of depth n and let $RT(\xi) = \eta$. We add immediate successors to ξ according to the following clauses:

- If $M_n^*(\xi) = CONE_\tau(\eta)$ then for every path π in MT leaving $CONE_\tau(\eta)$ we add to ξ an immediate successor ξ_1 in MT^* and we define $RT(\xi_1) = \eta_1$, where η_1 is the first node in π after π leaves $CONE_\tau(\eta)$ and $M_e^*(\xi, \xi_1) = M_e(\eta', \eta)$, where η' is the predecessor of η in π.
- If $M_n^*(\xi) = \{\eta\}$ where η is not a leaf in MT we add for every immediate successor η_1 of η an immediate successor ξ_1 to ξ. We define $RT(\xi_1) = \eta_1$ and $M_e^*(\xi, \xi_1) = M_e(\eta, \eta_1)$.
- If $M_n^*(\xi) = \{\eta\}$ where η is a leaf in MT then ξ is a leaf in MT^* and $RT(\xi) = \eta$.

Next, we define M_n^* for all nodes added in step $n+1$ of the induction. Let ξ be such a node. Two cases:

- If there is no τ-avoiding path starting from $RT(\xi)$ in MT for any $\tau \in P$ then $M_n^*(\xi) = \{RT(\xi)\}$.
- Otherwise, consider the set $\mathcal{S}$ of all transitions, τ, for which there is an infinite τ-avoiding path starting from $RT(\xi)$ in MT. Let τ_1 be the transition chosen least recently, possible not at all, as a $CONE$'s directive along the sequence $M_n^*(\xi_0), M_n^*(\xi_1), \ldots, M_n^*(\xi_{n-1})$, where $\xi_0\xi_1 \ldots \xi_{n-1}\xi$ is the path from the root to ξ in MT^*. We define $M_n^*(\xi) = CONE_{\tau_1}(RT(\xi))$. In the case there are more than one such transitions in $\mathcal{S}$ the transition with the smallest index (assume all transitions in P are indexed) is chosen.

Lemma A.1:

- For every node ξ in MT^*, $RT(\xi) \in M_n^*(\xi)$.
- For every two nodes ξ_1 and ξ_2 in MT^*, $M_n^*(\xi_1) \cap M_n^*(\xi_2) = \emptyset$.
- MT^* covers MT, i.e., every node of MT belongs to some $M_n^*(\xi)$, where ξ is a node of MT^*.

Proof:

- According to the definition of MT^*, $M_n^*(\xi)$ is either $\{RT(\xi)\}$ or $CONE_{\tau_1}(RT(\xi))$ in both cases $RT(\xi) \in M_n^*(\xi)$.
- According to the definition of MT^*, at every step of the induction $M_n^*(\xi)$ contains nodes of T that are not included in any previously defined $M_n^*(\xi')$. Moreover, if ξ_1 and ξ_2 are added to MT^* in the same induction step then $RT(\xi_1)$ and $RT(\xi_2)$ do not reside on a common path in T. Therefore according to the definition of M_n^* and the tree structure of T, $M_n^*(\xi_1) \cap M_n^*(\xi_2) = \emptyset$.
- According to the definition of MT^*, the root of MT^* covers the root of MT and in the induction step, given a node ξ, all immediate successors in T of nodes in $M_n^*(\xi)$ are covered.

Lemma A.2: The tree MT^* is well-founded, i.e., contains finite paths only.

Proof: Assume there exists an infinite path $\pi^* = \xi_0\xi_1\ldots$ in MT^*. According to the definition of the function RT the nodes $RT(\xi_0), RT(\xi_1), \ldots$ are nodes in MT. Moreover, according to the definition of MT^* there exists a path $\pi = \eta_0\eta_1\ldots$ in MT such that for every $i \geq 0$, $RT(\xi_{i+1})$ belongs to π and it appears in π after, but not necessarily immediately after, $RT(\xi_i)$. Since the sequence $RT(\xi_0), RT(\xi_1), \ldots$ is infinite, π is also infinite and thus, unfair.

Therefore, there are several (at lease one) transitions such that from some point on in π are continuously enabled but are never selected for execution. Let τ be such a transition with the smallest index and let $\pi_i = RT(\xi_i)\eta_k\eta_{k+1}\ldots$ be a suffix of π which is τ-avoiding and which starts from $RT(\xi_i)$.

Two possibilities:

- τ is not selected as a *CONE*'s directive along $M_n^*(\xi_0), M_n^*(\xi_1)\ldots, M_n^*(\xi_{i-1})$ then according to the definition of MT^*, $M_n^*(\xi_i) = CONE_\tau(RT(\xi_i))$.
- τ is selected as a *CONE*'s directive along $M_n^*(\xi_0), M_n^*(\xi_1)\ldots, M_n^*(\xi_{i-1})$ then according to the definition of MT^* there exists $j > i$ such that τ is least recently selected as a *CONE*'s directive along $M_n^*(\xi_0), M_n^*(\xi_1)\ldots, M_n^*(\xi_{j-1})$ and thus $M_n^*(\xi_j) = CONE_\tau(RT(\xi_j))$.

Let k denote i if the first case holds and j otherwise. In both cases, the infinite tail of π which is τ-avoiding is contained in $CONE_\tau(RT(\xi_k))$. Therefore, all the nodes $RT(\xi_{k+1}), RT(\xi_{k+2}), \ldots$ are contained in $M_n^*(\xi_k)$, a contradiction to Lemma B.1. $\Box$

Based on the properties of MT^* proved in Lemmas B.1 and B.2 we next continue the $\mathcal{AU}$-rule completeness proof. Both trees MT and MT^* are constructed for a specific initial state σ_0 that satisfies p, i.e., the root node of MT is mapped by M_n to σ_0. In order to get rid of the dependency of the trees on σ_0 we combine all trees MT such that their root node is mapped to a state satisfying p into an infinitary tree $\overline{MT} =< \overline{T}, \overline{M_n}, \overline{M_e} >$. A new root is added and its immediate successors are all the trees MT s.t. $M_n(\eta_0) \models p$. Similarity we combine all MT^* trees into an infinitary well-founded tree $\overline{MT^*} =< \overline{T^*}, \overline{M_n^*}, \overline{M_e^*} >$.

Next, the nodes of $\overline{MT^*}$ are ranked by countable ordinals. All leaves are ranked with 0, an intermediate node is ranked with the successor of the least upper bound of the ranks of its immediate successors. In order to rank nodes with a unique rank a *rank-shift* is performed. Let $\varrho(\xi)$ denote the rank defined for node ξ by the above procedure.

Let Q be the set of all nodes in $\overline{MT}$ that reside on finite paths and are nor leaves neither the root of $\overline{MT}$. We define Φ to be the assertion satisfied by exactly all states that mark the nodes in Q, that is:

$$\Phi = \{\sigma | \exists \eta : \ \sigma = M_n(\eta) \wedge \eta \in Q\}$$

We define the ranking function δ to be:

$$\delta(\sigma) = w \quad \textit{iff} \quad \exists B : B \neq \emptyset \wedge B \subseteq \overline{T^*} \wedge (\xi \in B \Leftrightarrow \sigma \in \overline{M_n}(\overline{M_n^*}(\xi))) \wedge w = min(\bigcup_{\xi \in B} \varrho(\xi))$$

Next we show that the premises $AU1. - AU4.$ hold for Φ and δ above.

- $AU1$. According to the construction of $\overline{MT}$ all states that satisfy p mark at least one node in $\overline{MT}$. Moreover, every state that satisfy p either mark a node in Q (thus it satisfies Φ) or a leaf of $\overline{MT}$ (recall that every leaf η in $\overline{MT}$ satisfies $\overline{M_n}(\eta) \models I_2$) therefore

$$\models p \rightarrow \Phi \vee I_2$$

Moreover, $\overline{MT^*}$ covers $\overline{MT}$ and therefore the ranking function δ is defined for every state that mark a node in $\overline{MT}$. In particular, all states in p mark nodes in $\overline{MT}$ and thus

$$\models p \rightarrow (\delta \in \mathcal{W})$$

- $AU2$. From every state σ that satisfies Φ and $\delta(\sigma) = w$ the execution of any transition from p leads to a state that mark either a leaf of $\overline{MT}$ and therefore I_2 is satisfied or it leads to a state that mark an internal node (not a leaf) η of $\overline{MT}$. Since every finite prefix of a path can be extended to a fair path $\eta \in Q$ and therefore $\overline{M_n}(\eta) \models \Phi$. Thus,

$$\{\Phi \wedge (\delta = w)\}P\{I_2 \vee \Phi\}$$

Moreover, according to the definition of $\overline{MT^*}$ if σ marks ξ in $\overline{T^*}$ then any transition of P leads to a state that either mark ξ or mark an immediate successor of ξ and therefore

$$\{\Phi \wedge (\delta = w)\}P\{(\delta \leq w)\}$$

- $AU3$. According to the construction of $\overline{MT}$ and the definition of Q all nodes in Q are marked by states satisfying I_1 therefore

$$\models \Phi \rightarrow I_1$$

- $AU4$. Given $w \in \mathcal{W}$ if there does not exist a state σ such that $\sigma \models \Phi$ and $\delta(\sigma) = w$ then $\models \Phi \wedge (\delta = w) \leftrightarrow$ *false* and therefore both conditions in $AU4$. hold vacuously.
In there exists a state σ such that $\sigma \models \Phi$ and $\delta(\sigma) = w$ then $\exists \xi \in \overline{MT^*}$ such that $\varrho(\xi) = w$ and $\sigma \in \overline{M_n}(\overline{M_n^*}(\xi))$. Consider two cases:
Case 1: $\overline{M_n^*}(\xi) = CONE_\tau(RT(\xi))$.
According to the definition of $CONE_\tau$, τ is enabled in all states σ' such that $\sigma' \in \overline{M_n}(CONE_\tau(RT(\xi))$ therefore $\sigma \models En(\tau)$ and we conclude

$$\models (\Phi \wedge (\delta = w)) \rightarrow En(\tau)$$

According to the definition of $CONE_\tau$ every τ-move leaves $CONE_\tau(RT(\xi))$ and therefore a τ-move reaches an immediate successor ξ' of ξ in $\overline{MT^*}$. Since the nodes in $\overline{MT^*}$ are ranked leaves up we know $\varrho(\xi') < \varrho(\xi)$. Thus,

$$\Phi \wedge (\delta = w)\}\tau\{\delta < w\}$$

Moreover, if the τ-move reaches a leaf of $\overline{MT}$ then the resulting state satisfies I_2 otherwise it satisfies Φ. Thus,

$$\{\Phi \wedge (\delta = w)\}\tau\{I_2 \vee \Phi\}$$

Case 2: $\overline{M_n^*}(\xi) = RT(\xi)$.

In this case, $\sigma = \overline{M_n}(RT(\xi))$. $\sigma \models \Phi$ implies that $\exists \eta : \sigma = \overline{M_n}(\eta) \wedge \eta \in Q$ therefore η and $RT(\xi)$ are roots to identical subtrees of $\overline{MT}$ and thus $RT(\xi) \in Q$. Thus, $RT(\xi)$ is not a leaf in $\overline{MT}$ and there exists τ such that $\sigma \models En(\tau)$ and we conclude

$$\models (\Phi \wedge (\delta = w)) \rightarrow En(\tau)$$

Moreover, the τ-move reaches an immediate successor ξ' of ξ in $\overline{MT^*}$ and we know $\varrho(\xi') < \varrho(\xi)$:

$$\{\Phi \wedge (\delta = w)\}\tau\{\delta < w\}$$

If the τ-move reaches a leaf of $\overline{MT}$ then the resulting state satisfies I_2 otherwise it satisfies Φ. Thus,

$$\{\Phi \wedge (\delta = w)\}\tau\{I_2 \vee \Phi\}$$

$\dfrac{p \rightarrow q}{P\ \mathit{Sat}\ p \rightarrow q}$ For $q \in L$	$\dfrac{p \rightarrow \neg q}{P\ \mathit{Sat}\ p \rightarrow \neg q}$ For $q \in L$
$\dfrac{\begin{array}{l} P\ \mathit{Sat}\ p \rightarrow f_1 \\ P\ \mathit{Sat}\ p \rightarrow f_2 \end{array}}{P\ \mathit{Sat}\ p \rightarrow (f_1 \wedge f_2)}$ $\quad$ $\dfrac{\begin{array}{l} P\ \mathit{Sat}\ p_1 \rightarrow f \\ P\ \mathit{Sat}\ p_2 \rightarrow f \end{array}}{P\ \mathit{Sat}\ (p_1 \vee p_2) \rightarrow f}$	$\dfrac{P\ \mathit{Sat}\ p \rightarrow \neg f_1}{P\ \mathit{Sat}\ p \rightarrow \neg(f_1 \wedge f_2)}$
$\dfrac{\begin{array}{l} P\ \mathit{Sat}\ q \rightarrow f_1 \\ \{p\}P\{q\} \end{array}}{P\ \mathit{Sat}\ p \rightarrow AX f_1}$	$\dfrac{P\ \mathit{Sat}\ p \rightarrow EX \neg f_1}{P\ \mathit{Sat}\ p \rightarrow \neg AX f_1}$
$\dfrac{\begin{array}{l} P\ \mathit{Sat}\ q \rightarrow f_1 \\ \text{There exists } \tau : \{p\}\tau\{q\} \text{ and } p \rightarrow En(\tau) \end{array}}{P\ \mathit{Sat}\ p \rightarrow EX f_1}$	$\dfrac{P\ \mathit{Sat}\ p \rightarrow AX \neg f_1}{P\ \mathit{Sat}\ p \rightarrow \neg EX f_1}$
$\dfrac{\begin{array}{l} P\ \mathit{Sat}\ I_1 \rightarrow f_1 \\ P\ \mathit{Sat}\ I_2 \rightarrow f_2 \\ P\ \mathit{Sat}\ p \rightarrow A[I_1 \mathcal{U} I_2] \end{array}}{\begin{array}{l} P\ \mathit{Sat}\ p \rightarrow A[f_1 \mathcal{U} f_2] \\ \text{where, } f_1 \notin L \text{ or } f_2 \notin L. \end{array}}$ ********************** $\dfrac{AU1 - AU4}{\begin{array}{l} P\ \mathit{Sat}\ p \rightarrow A[I_1 \mathcal{U} I_2] \\ \text{where, } I_1, I_2 \in L. \end{array}}$	$\dfrac{\begin{array}{l} P\ \mathit{Sat}\ p \rightarrow EG \neg f_2 \\ \text{or} \\ P\ \mathit{Sat}\ p \rightarrow E[(\neg f_2) \mathcal{U} (\neg f_1 \wedge \neg f_2)] \end{array}}{P\ \mathit{Sat}\ p \rightarrow \neg A[f_1 \mathcal{U} f_2]}$
$\dfrac{\begin{array}{l} P\ \mathit{Sat}\ I_1 \rightarrow f_1 \\ P\ \mathit{Sat}\ I_2 \rightarrow f_2 \\ P\ \mathit{Sat}\ p \rightarrow E[I_1 \mathcal{U} I_2] \end{array}}{\begin{array}{l} P\ \mathit{Sat}\ p \rightarrow E[f_1 \mathcal{U} f_2] \\ \text{where, } f_1 \notin L \text{ or } f_2 \notin L. \end{array}}$ ********************** $\dfrac{AU1, AU3, AU4}{\begin{array}{l} P\ \mathit{Sat}\ p \rightarrow E[I_1 \mathcal{U} I_2] \\ \text{where, } I_1, I_2 \in L. \end{array}}$	$\dfrac{\begin{array}{l} P\ \mathit{Sat}\ p \rightarrow (\neg f_1 \wedge \neg f_2) \\ \text{or} \\ p \rightarrow I \\ P\ \mathit{Sat}\ I \rightarrow \neg f_2 \wedge AX(I \vee (\neg f_1 \wedge \neg f_2)) \end{array}}{P\ \mathit{Sat}\ p \rightarrow \neg E[f_1 \mathcal{U} f_2]}$
$\dfrac{P\ \mathit{Sat}\ p \rightarrow \neg E[\mathit{true}\ \mathcal{U} \neg f_1]}{P\ \mathit{Sat}\ p \rightarrow AG f_1}$	$\dfrac{P\ \mathit{Sat}\ p \rightarrow E[\mathit{true}\ \mathcal{U} \neg f_1]}{P\ \mathit{Sat}\ p \rightarrow \neg AG f_1}$
$\dfrac{EG1. - EG6.}{P\ \mathit{Sat}\ p \rightarrow EG f_1}$	$\dfrac{P\ \mathit{Sat}\ p \rightarrow A[\mathit{true}\ \mathcal{U} \neg f_1]}{P\ \mathit{Sat}\ p \rightarrow \neg EG f_1}$

Figure 1: The deduction system