An Axiomatization of Lamport's Temporal Logic of Actions

Martín Abadi

October 12, 1990
revised March 4, 1993

A preliminary version of this report appeared in the proceedings of the CONCUR '90 conference, held in Amsterdam, The Netherlands, in August 1990 [BK90].

Author's Abstract

Lamport recently invented a temporal logic of actions suitable for expressing concurrent programs and for reasoning about their computations. In this logic, actions have syntactic representations, which can be combined and analyzed. The basic construct for relating actions and computations is $[\;]$; a computation satisfies the formula $[A]$ if either the computation has halted or the first action in the computation is an A action. In addition, the language includes the temporal operators $\Box$ ("always") and $\Diamond$ ("eventually"), and thus it is easy to write both safety and liveness formulas.

However, the temporal logic of actions is not very expressive in some respects (just expressive enough). One cannot define the "next" and the "until" operators of many previous temporal logics. This is actually a feature, in that formulas with "until" are too often incomprehensible, and "next" violates the important principle of invariance under stuttering.

A proof system for the logic of actions might be obtained by translating into previous, richer formalisms. In this translation we forfeit the logic and its advantages. A new suit of rules for temporal reasoning with actions is therefore wanted. A complete axiomatization can provide some guidance in choosing and understanding the rules used in practice, and in particular the laws for reasoning about programs.

In this paper, we study a proof system for a propositional logic, PTLA. After an informal introduction, we define the syntax and semantics of PTLA precisely, and then present our proof system and prove its completeness.

Contents

1 Introduction **1**

2 An Example **2**

3 The Syntax and Semantics of PTLA **5**

3.1 Syntax 5

3.2 Semantics 6

4 A Complete Proof System **7**

4.1 The System 7

4.2 Some Consequences of the Axioms 9

4.3 Soundness and Completeness 9

5 Conclusions **16**

Acknowledgements **16**

References **17**

1 Introduction

Lamport recently invented a temporal logic of actions suitable for expressing concurrent programs and for reasoning about their computations [Lam90]. In this logic, actions have syntactic representations, which can be combined and analyzed. Lamport views an action as a state transition, and a computation as a sequence of states. The basic construct for relating actions and computations is $[\ ]$; a computation satisfies the formula $[A]$ if either the computation has halted or the first action in the computation is an A action. The dual notation is $\langle A \rangle$, which means that the computation has not halted and that the first action is an A action. (Notice that $[A]$ and $\langle A \rangle$ are formulas, and not modalities as in dynamic logic [Pra76] and in Hennessy-Milner logic [HM85].) In addition, the language includes the temporal operators $\Box$ ("always") and $\Diamond$ ("eventually"), and thus it is easy to write both safety and liveness properties [Pnu77].

However, the temporal logic of actions is not very expressive in some respects (just expressive enough). One cannot define the "next" and the "until" operators of many previous temporal logics [Pnu81]. This is actually deliberate; formulas with nested occurrences of "until" are too often incomprehensible, and "next" violates the principle of invariance under stuttering, which is important for hierarchical and compositional reasoning [Lam89].

A proof system for the logic of actions might be obtained by translating into previous, richer formalisms. In this translation we forfeit the logic and two of its main advantages, understandable formulas and the possibility of reducing many arguments to simple calculations on actions. A new suit of rules for temporal reasoning with actions is therefore wanted. A complete axiomatization can provide some guidance in choosing and understanding the rules used in practice, and in particular the laws for reasoning about programs. (A decision procedure is less often helpful in this respect.)

At least two kinds of complete proof systems are possible: a propositional system and a first-order system. (In the first-order case, one can hope only for relative or nonstandard completeness results, of course.) In this paper, we study a proof system for a propositional temporal logic of actions, PTLA.

In the next section, we introduce the logic of actions through an example and discuss the underlying model very informally. We give precise definitions of the syntax and semantics of PTLA in Section 3. In Section 4, we present our proof system and prove its completeness.

2 An Example

In this example, we use the logic of actions for describing a trivial complete program with two processes. Process S repeatedly sends a boolean to process R and then waits for a signal; process R repeatedly receives a value from process S and then signals.

In a CSP-like notation, this program is:

$$[S :: *[\,[R\,!\,\mathit{true} \,\Box\, R\,!\,\mathit{false}]\ ;\ R\,?\,ANY]\,] \quad || \quad [R :: *[S\,?\,x\ ; S\ !\,ANY]\,]$$

(Here ANY is used as in Occam, for synchronization without message-passing [INM84][1].)

We take a program, such as this one, to denote the set of behaviors that it generates. In the logic of actions, a behavior is a sequence of states. It may help to view a state as a snapshot of a device that executes the program. All that matters is that each program variable has a value at each state.

Formally, behaviors are described in terms of actions. An action is a binary relation on program states. Intuitively, an action is the set of all pairs of states s and t such that the action can change the state from s to t; an action is enabled in s if it can change the state from s to t for some t. A predicate on primed and unprimed program variables expresses an action. For example, the action that negates the value of a variable y may be written $y' \equiv \neg y$ (or, equivalently, $y \equiv \neg y'$, or $(y \wedge \neg y') \vee (\neg y \wedge y')$); y' represents the value of y after the action.

In the semantics, then, states are primitive, and actions are not; this presents advantages and disadvantages in comparison with models with primitive actions. At any rate, the two approaches are valid, and they can provide the same sort of information (as one can translate between sequences of states and sequences of events). The properties of programs discussed in the logic of actions are interesting in either model.

Coming back to our example, we start the formal description of the program by listing its variables. In addition to the variable x, the program

[1]Occam is a trade mark of the INMOS Group of Companies.

has two implicit control variables l_S and l_R, one for each process; a device executing the program would keep track of the values of these three variables. The boolean l_S is true when control is at the send command in process S, and false when control is at the receive command in process S. The boolean l_R is true when control is at the receive command in process R, and false when control is at the send command in process R. Thus, a state should assign values to x, l_S, and l_R.

Next we define an action for each communication event. The sending of *true* can change a state where l_S and l_R hold to a state where $\neg l_S$, $\neg l_R$, and x hold. Hence, for the sending of *true*, we write:[2]

$$A_t \triangleq l_S \wedge \neg l'_S \wedge l_R \wedge \neg l'_R \wedge x'$$

A similar formula expresses the sending of *false*:

$$A_f \triangleq l_S \wedge \neg l'_S \wedge l_R \wedge \neg l'_R \wedge \neg x'$$

The nondeterministic composition of these two actions is represented as a disjunction:

$$A \triangleq A_t \vee A_f$$

The other basic action of the program is the acknowledgement:

$$Ack \triangleq \neg l_S \wedge l'_S \wedge \neg l_R \wedge l'_R \wedge (x' \equiv x)$$

Thanks to the use of control variables, disjunction represents sequential composition, in addition to nondeterministic composition.[3] Thus, the sequential composition of A and Ack is represented as a disjunction:

$$N \triangleq A \vee Ack$$

The action N is the next-state relation of the complete program. A computation of the program, started from an arbitrary state, satisfies $\Box[N]$.

The program is enabled (that is, it can make progress) only when l_S and l_R have the same value. We define:

$$Enabled(N) \triangleq (l_S \equiv l_R)$$

[2] The symbol $\triangleq$ means *equals by definition*.

[3] In Lamport's interleaving model, the action that corresponds to the parallel composition of two processes is the union of the actions that correspond to the processes, so disjunction represents parallel composition as well.

The formula $\Box[N]$ allows some computations that are immediately deadlocked, when started in a state where $\mathit{Enabled}(N)$ does not hold. To restrict attention to the computations that start from the expected initial states, we define the predicate *Init*:

$$\mathit{Init} \triangleq l_S \wedge l_R$$

A computation satisfies $\mathit{Init} \wedge \Box[N]$ if it is a computation of the program that starts in a state where *Init* holds. One can prove formally that none of these computations deadlocks:

$$\mathit{Init} \wedge \Box[N] \Rightarrow \Box \mathit{Enabled}(N)$$

It is still possible for a computation that satisfies *Init* and $\Box[N]$ to halt, because we have not yet made liveness assumptions (both *Init* and $\Box[N]$ are safety formulas). The assumption of weak fairness for N suffices to guarantee continued progress. Weak fairness for N says that if N is always enabled after a certain point then eventually N takes place:

$$\mathrm{WF}(N) \triangleq \Box(\Box \mathit{Enabled}(N) \Rightarrow \Diamond\langle N\rangle)$$

The desired progress property follows:

$$(\mathit{Init} \wedge \Box[N] \wedge \mathrm{WF}(N)) \Rightarrow (\Box\Diamond l_R \wedge \Box\Diamond\neg l_R \wedge \Box\Diamond l_S \wedge \Box\Diamond\neg l_S)$$

A further requirement is that *true* and *false* are chosen fairly. Strong fairness for A_t says that if the transmission of *true* is enabled infinitely often (that is, $l_S \wedge l_R$ holds infinitely often), then the transmission of *true* happens eventually. Hence we set:

$$\begin{array}{rcl} \mathit{Enabled}(A_t) & \triangleq & l_S \wedge l_R \\ \mathrm{SF}(A_t) & \triangleq & \Box(\Box\Diamond \mathit{Enabled}(A_t) \Rightarrow \Diamond\langle A_t\rangle) \end{array}$$

and strong fairness for A_f is written similarly:

$$\begin{array}{rcl} \mathit{Enabled}(A_f) & \triangleq & l_S \wedge l_R \\ \mathrm{SF}(A_f) & \triangleq & \Box(\Box\Diamond \mathit{Enabled}(A_f) \Rightarrow \Diamond\langle A_f\rangle) \end{array}$$

Under these strong-fairness assumptions, the program guarantees that the value of x is infinitely often *true* and infinitely often *false*:

$$(\mathit{Init} \wedge \Box[N] \wedge \mathrm{WF}(N) \wedge \mathrm{SF}(A_t) \wedge \mathrm{SF}(A_f)) \Rightarrow (\Box\Diamond x \wedge \Box\Diamond\neg x)$$

In introducing the logic through this example we have only exercised the notation, and not reasoned within it formally. The traditional approaches to safety and liveness verification are adequate for proving the properties that we have claimed in the example. Lamport has formalized these traditional approaches within the logic, and has exploited them in the study of moderately substantial algorithms (in particular with the general logic, mentioned in Section 5).

3 The Syntax and Semantics of PTLA

In this section, we give a precise definition of the syntax and semantics of a propositional temporal logic of actions, PTLA. This logic, although not introduced in [Lam90], is a formalization of Lamport's approach in a propositional setting.

3.1 Syntax

We have a countably infinite collection of *proposition symbols* $P_0, P_1, P_2, \ldots$ and a countably infinite collection of *action symbols* $A_0, A_1, A_2, \ldots$.

A *state predicate* is a boolean combination of proposition symbols. (We use the boolean connectives *false*, $\neg$, and $\wedge$, and view the connectives $\vee$, $\Rightarrow$, and $\equiv$ as abbreviations.) If P is a state predicate, then P' is a *primed state predicate*. An *action* is a boolean combination of state predicates, primed state predicates, and action symbols; thus, in particular, a state predicate is an action. This repertoire of actions is richer than that allowed in Hennessy-Milner logic (where only action symbols are considered); on the other hand, the regular expressions and the context-free grammars of dynamic logic do not seem necessary here.

A *formula* of the logic is:

- a state predicate;
- $[A]$, where A is an action;
- a boolean combination of formulas; or
- $\Box F$, where F is a formula.

We also write $\langle A \rangle$ for $\neg[\neg A]$, and $\Diamond F$ for $\neg\Box\neg F$.

Throughout, we use the letters O, P, Q, and R for state predicates, A and B for actions, and F and G for arbitrary formulas.

Lamport's logic also includes action formulas, for example of the form $A = B$. For simplicity, we do not allow these formulas, as it is possible to use paraphrases, such as $\Box([A] = [B])$ for $A = B$.

The primitive action symbols A_0, A_1, A_2, ... were not needed in the example of Section 2, and hence some motivation for them is in order. Often an action cannot be expressed as a boolean combination of state predicates and primed state predicates, because we have not been given a full specification of the action, or because the action is essentially first-order, as $x' = x + 1$. In these cases, having action symbols enables us to name the action and exploit any known propositional facts about it. For example, if A_0 stands for $x' = x + 1$, P_0 for $x = 0$, and P_1 for $x = 1$, then P_1' is $x' = 1$, and we can write and use $P_0 \wedge A_0 \Rightarrow P_1'$; alternatively, $\Box[P_0 \wedge A_0 \Rightarrow P_1']$ achieves the same effect.

3.2 Semantics

The semantics of the temporal logic of actions resembles those for other linear-time temporal logics. The novelties concern the meaning of the formulas of the form $[A]$.

An interpretation is a pair $(\mathbf{S}, I)$ where

- $\mathbf{S}$ is a non-empty set; an element of $\mathbf{S}$ is called a *state*, and $\mathbf{S}$ is called a *state space*;
- I is a pair of mappings I_ρ and I_α, which assign to each proposition symbol a subset of $\mathbf{S}$ and to each action symbol a subset of $\mathbf{S} \times \mathbf{S}$, respectively; intuitively, $I_\rho(P_i)$ is the set of states where P_i is true, and $I_\alpha(A_i)$ is the set of pairs of states related by A_i.

Sometimes we omit mention of $\mathbf{S}$, and simply refer to I as the interpretation.

We extend the mapping I_ρ to all state predicates, by setting:

$$\begin{array}{rcl} I_\rho(\mathit{false}) & \triangleq & \emptyset \\ I_\rho(\neg P) & \triangleq & \mathbf{S} - I_\rho(P) \\ I_\rho(P \wedge Q) & \triangleq & I_\rho(P) \cap I_\rho(Q) \end{array}$$

Then we extend the mapping I_α to all actions, by setting:

$$I_\alpha(P) \triangleq I_\rho(P) \times \mathbf{S}$$

$$
\begin{aligned}
I_\alpha(P') &\triangleq \mathbf{S} \times I_\rho(P) \\
I_\alpha(\neg A) &\triangleq \mathbf{S} \times \mathbf{S} - I_\alpha(A) \\
I_\alpha(A \wedge B) &\triangleq I_\alpha(A) \cap I_\alpha(B)
\end{aligned}
$$

A *behavior* over $\mathbf{S}$ is an infinite sequence of elements of $\mathbf{S}$. If σ is the behavior $s_0, s_1, s_2, \ldots$, we denote s_i by σ_i and $s_i, s_{i+1}, s_{i+2}, \ldots$ by σ^{+i}. We say that σ *is halted* if $\sigma_0 = \sigma_i$ for all i. If σ is not halted, we write $\mu(\sigma)$ for the least i such that $\sigma_0 \neq \sigma_i$.

A *model* is a triple $(\mathbf{S}, I, \sigma)$, where $(\mathbf{S}, I)$ is an interpretation and σ is a behavior over $\mathbf{S}$. We define the *satisfaction relation* between models and formulas inductively, as follows:

$$
\begin{aligned}
(\mathbf{S}, I, \sigma) \models P_j &\triangleq \sigma_0 \in I_\rho(P_j) \\
(\mathbf{S}, I, \sigma) \models [A] &\triangleq \text{either } \sigma \text{ is halted or } (\sigma_0, \sigma_{\mu(\sigma)}) \in I_\alpha(A) \\
(\mathbf{S}, I, \sigma) \models \mathit{false} &\triangleq \text{false} \\
(\mathbf{S}, I, \sigma) \models \neg F &\triangleq (\mathbf{S}, I, \sigma) \not\models F \\
(\mathbf{S}, I, \sigma) \models F \wedge G &\triangleq (\mathbf{S}, I, \sigma) \models F \text{ and } (\mathbf{S}, I, \sigma) \models G \\
(\mathbf{S}, I, \sigma) \models \Box F &\triangleq \text{for all } i, (\mathbf{S}, I, \sigma^{+i}) \models F
\end{aligned}
$$

For example, $(\mathbf{S}, I, \sigma) \models [\mathit{false}]$ if and only if σ is halted.

The formula F is *satisfiable* if there exist $(\mathbf{S}, I, \sigma)$ such that $(\mathbf{S}, I, \sigma) \models F$. The formula F is *valid* if $(\mathbf{S}, I, \sigma) \models F$ for all $(\mathbf{S}, I, \sigma)$; we write this $\models F$.

4 A Complete Proof System

In the first subsection we give our axioms, and then we list some of their consequences. Finally, we prove the completeness of the axioms.

4.1 The System

The temporal logic of actions is an extension of the common temporal logic with the single modality $\Box$. Accordingly, we are going to base our axiomatization on a usual one, a system known as D (in [HC68]) or S4.3*Dum* (in [Gol87]). The axioms and rules for D are:

1. $\vdash \Box(F \Rightarrow G) \Rightarrow (\Box F \Rightarrow \Box G)$

2. $\vdash \Box F \Rightarrow F$

3. $\vdash \Box F \Rightarrow \Box\Box F$

4. $\vdash \Box(\Box F \Rightarrow G) \vee \Box(\Box G \Rightarrow F)$

5. $\vdash \Box(\Box(F \Rightarrow \Box F) \Rightarrow F) \Rightarrow (\Diamond\Box F \Rightarrow F)$

6. If $\vdash F$ then $\vdash \Box F$.

7. If F is an instance of a propositional tautology then $\vdash F$.

8. If $\vdash F$ and $\vdash F \Rightarrow G$ then $\vdash G$.

Axiom 4 is a classical way to express that time is linear—that any two instants in the future are ordered. Axiom 5, indirectly attributed to Geach in [HC68], is a simplification of the original $\Box(\Box(F \Rightarrow \Box F) \Rightarrow F) \Rightarrow (\Diamond\Box F \Rightarrow \Box F)$, due to Dummett and Lemmon; Axiom 5 expresses the discreteness of time.

We introduce some axioms about actions:

9. $\vdash [\mathit{false}] \Rightarrow [A]$

10. $\vdash \neg[\mathit{false}] \Rightarrow [P] \equiv P$

11. $\vdash \neg[\mathit{false}] \Rightarrow [\neg A] \equiv \neg[A]$

12. $\vdash [A \wedge B] \equiv [A] \wedge [B]$

13. $\vdash [(\neg P)'] \equiv [\neg P']$

14. $\vdash [(P \wedge Q)'] \equiv [P' \wedge Q']$

15. $\vdash \Box P \Rightarrow [P']$

16. $\vdash \Box(P \Rightarrow (([P'] \wedge G) \vee \Box G)) \Rightarrow (([P'] \wedge G) \Rightarrow \Box G)$

Axiom 16 can be paraphrased as follows: suppose that whenever P holds either G holds and P survives the next state change, or G is true forever; thus, G holds for as long as P holds, and becomes true forever if P stops holding; hence, if G is true initially and P is true after the first state change then G is always true. All the other axioms are rather straightforward.

Our axiomatization could perhaps be simplified. It is worth recalling, however, that a less expressive logic does not always have a simpler proof system. For instance, the system for temporal logic with "next" is simpler than D [GPSS80], yet the "next" modality increases the expressiveness of the logic and its complexity (from coNP-complete to PSPACE-complete [SC85]).

4.2 Some Consequences of the Axioms

Some interesting consequences of the axioms are important in our completeness proof. We list and explain a few here.

- $\vdash [\mathit{false}] \Rightarrow (F \equiv \Box F) \wedge (F \equiv \Diamond F)$
  The formula expresses that once the computation has halted, all facts are permanent, meaning that F, $\Box F$, and $\Diamond F$ are equivalent for all F.

- $\vdash \langle P' \rangle \Rightarrow \Diamond P$
  In words, if the computation has not halted and the next action is P', then P holds eventually. This formula embodies the simplest method for proving liveness properties.

- $\vdash P \wedge \Box(P \Rightarrow [P']) \Rightarrow \Box P$
  This predictable induction principle follows directly from Axiom 16, when we instantiate G to P.

- $\vdash [P'] \wedge \Diamond G \Rightarrow G \vee \Diamond(P \wedge \Diamond G)$
  The theorem is another consequence of Axiom 16. It says if the action P' is about to take place (unless the computation halts) and G must hold eventually, then either G is true now, or P holds eventually and G holds later.

- $\vdash (P \wedge \Diamond G \wedge \Box(P \wedge \Diamond G \Rightarrow [P'])) \Rightarrow \Diamond(P \wedge G)$
  This is the dual to Axiom 16.

4.3 Soundness and Completeness

Theorem 1 (Soundness and Completeness) $\models F \;\Leftrightarrow\; \vdash F$

A simple induction on proofs shows that if $\vdash F$ then $\models \Box F$. It follows that if $\vdash F$ then $\models F$; thus, $\vdash$ is sound. The other direction of the claim (completeness) is more delicate, and the rest of this section is devoted to it.

Before embarking on the proof, we should recall a classical completeness theorem for D. The theorem says that if a formula G is not provable then $\neg G$ has a model. In fact, a model can be obtained from a structure of a very special form, known as a *balloon*. A balloon consists of a sequence of states $s_0, s_1, s_2, \ldots, s_m$, the *string*, and a set of states $\{t_0, t_1, t_2, \ldots, t_n\}$, the *bag*. (Without loss of generality, the states can be taken to be all distinct.) An interpretation gives values over these states to all the proposition symbols in G. With this interpretation, any sequence

$$s_0, \ldots, s_m, t_{i_0}, \ldots, t_{i_k}, \ldots$$

provides a model for $\neg G$, if all of $0, \ldots, n$ occur in $i_0, \ldots, i_k, \ldots$ infinitely often. Thus, a model is obtained from the balloon by linearizing the bag in any way whatsoever. This and similar constructions appear in [Gol87].

A formula *holds at a state* s in a behavior if it holds in the suffixes of the behavior that start with s. A formula *holds at a state* s in a balloon if it holds at s in all linearizations of the balloon. In both cases, we may also say that s *satisfies* the formula.

The basic strategy of our completeness proof is as follows. For every G, let G^* be the formula obtained from G by replacing each subformula of the form $[A]$ with a fresh proposition symbol; thus, $(\ )^*$ is a translation into a classical temporal formalism, with no actions. Assume that $\not\vdash F$; we want to show that $\not\models F$. That is, we want to find a model $(\mathbf{S}, I, \sigma)$ that satisfies $\neg F$. If $\not\vdash F$ then $\not\vdash (X \Rightarrow F)$, where X is any formula provable in PTLA (we will specify the choice of X below). A fortiori, $(X \Rightarrow F)$ cannot be derived using only the D axioms; because D does not have axioms about actions, it follows that $(X \Rightarrow F)^*$ cannot be derived in D. Thus, by the completeness of D, there must be a balloon $\mathcal{B}$ and an interpretation that satisfy $\neg(X \Rightarrow F)^*$. Obviously, X^* and $\neg F^*$ are also satisfied. The balloon and the interpretation will be useful in constructing a model for $\neg F$.

It is straightforward to choose X so that the proposition symbols that occur in F also occur in $\neg(X \Rightarrow F)^*$; hence the interpretation that satisfies $\neg(X \Rightarrow F)^*$ over $\mathcal{B}$ must assign truth values to these proposition symbols. Naturally, if the proposition symbol $[A]^*$ is mentioned in our completeness argument then it will occur in X^* (otherwise we could not say much about $[A]^*$); therefore, the interpretation must also assign a truth value to $[A]^*$. These properties of the interpretation will serve in defining the desired I.

In the course of the proof, we rely on the fact that each state in $\mathcal{B}$ satisfies certain theorems of PTLA, or rather their translation under $(\ )^*$. The number of theorems needed is finite, and their choice depends only on the choice of F. (It suffices to consider instances of Axioms 9 to 16 for subexpressions of $\neg F$, and some simple boolean combinations of these, sometimes with primes.) We take for X the conjunction of all formulas $\Box T$, where T is one of these necessary theorems. For all practical purposes, from now on, we may pretend that we have all the theorems of PTLA at our disposal.

For example, the proof of Proposition 1 uses that if P_j occurs in F then one of $[P_j']^*$ and $[(\neg P_j)']^*$ holds in each state in $\mathcal{B}$. To justify this claim, we point out that $\vdash [P_j'] \vee [(\neg P_j)']$, and we implicitly include $\Box([P_j'] \vee [(\neg P_j)'])$ in X. Since $\mathcal{B}$ satisfies X^*, every state in $\mathcal{B}$ satisfies $([P_j'] \vee [(\neg P_j)'])^*$, and hence one of $[P_j']^*$ and $[(\neg P_j)']^*$.

After these preliminaries, we are ready to start the necessary model construction.

Let $P_0, \ldots, P_k$ be a list of all the proposition symbols that occur in F. An *assignment* is a conjunction $Q_0 \wedge \ldots \wedge Q_k$ such that each Q_i is either P_i or $\neg P_i$. Given a state s in $\mathcal{B}$, there exists a unique assignment O_s that holds in s.

A state of $\mathcal{B}$ that satisfies $[\mathit{false}]^*$ is called a *halting* state. We have:

Proposition 1 *For every state s there exists an assignment R_s such that $[R_s']^*$ holds in s. Moreover, R_s is unique if and only if s is not a halting state.*

Proof To prove the existence of R_s, we first notice that $\vdash [P_j'] \vee [(\neg P_j)']$ for each proposition symbol P_j, and hence s must satisfy at least one of $[P_j']^*$ and $[(\neg P_j)']^*$. Therefore, let Q_j be one of P_j and $\neg P_j$ such that $[Q_j']^*$ holds at s. If R_s is the conjunction of all these Q_j's, then $[R_s']^*$ holds at s, because the axioms yield $\vdash [Q_0'] \wedge \ldots \wedge [Q_k'] \equiv [(Q_0 \wedge \ldots \wedge Q_k)']$. If s is not halting then R_s is unique because $\vdash \neg[\mathit{false}] \Rightarrow [\neg P_i'] \equiv \neg[P_i']$ and $\vdash [\neg P_i'] \equiv [(\neg P_i)']$. If s is a halting state then R_s could be any assignment, since both $\vdash [\mathit{false}] \Rightarrow [P_i']$ and $\vdash [\mathit{false}] \Rightarrow [(\neg P_i)']$. ∎

Given a state s which is not halting, we say that t *follows* s in $\mathcal{B}$ if:

- t is the first state in the string strictly after s such that $O_t = R_s$, if such exists;
- else, t is a state in the bag and $O_t = R_s$, if such exists;
- else, $t = s$ if $O_s = R_s$.

Proposition 2 *If s is not halting, then some t follows s.*

Proof The axioms yield $\vdash [R_s'] \wedge \neg[\mathit{false}] \Rightarrow \Diamond R_s$, so s must satisfy this formula. Thus, R_s must hold at a state in the string strictly after s, or at a state in the bag, or at s itself. ∎

We proceed to construct the desired behavior σ inductively.

- Let σ_0 be the first state in the string of the balloon, or an arbitrary state in the bag if the string is empty.

- If σ_i is a halting state, let $\sigma_{i+1} = \sigma_i$.

- If σ_i is not a halting state, let σ_{i+1} be a state that follows σ_i. If possible, always pick a state that has not been visited previously. Otherwise, pick the state first visited the longest time ago, and start cycling.

The behavior σ ends with a cycle. It may be that this cycle is of length one while the single state s in the cycle does not satisfy $[\mathit{false}]^*$. In this case, we modify σ trivially: we make a copy $\hat{s}$ of s and have σ cycle between these two different states. (Of course, s and $\hat{s}$ are different only formally, as they satisfy the same formulas.) This modification is convenient in giving a proper meaning to $[\mathit{false}]$. We do not discuss this minor point in the construction further, and leave the obvious details to the reader.

Let $\mathcal{S}$ be the balloon obtained by discarding from $\mathcal{B}$ all states not in σ. More precisely, the bag of $\mathcal{S}$ is the set of states that occur in the cyclic part of σ, and the string of $\mathcal{S}$ is the remaining states of σ, ordered in the order of their occurrence. The bag of $\mathcal{S}$ may be a subset of the bag of $\mathcal{B}$. It may happen, however, that the bag of $\mathcal{S}$ consists of a single state s from the string of $\mathcal{B}$ (or s and $\hat{s}$); this is in the case where either s is halting or s is the last state to satisfy R_s.

Next we show that $\neg F^*$ still holds in $\mathcal{S}$. We strengthen the claim, to mention every state and every subformula of $\neg F^*$. However, it is not claimed that all of the theorems compiled in X still hold at each state in $\mathcal{S}$; this claim is not needed.

Proposition 3 *If G^* is a subformula of $\neg F^*$ and s is a state in $\mathcal{S}$, then G^* holds at s in $\mathcal{S}$ if and only if G^* holds at s in $\mathcal{B}$. In particular, all linearizations of $\mathcal{S}$ satisfy $\neg F^*$.*

Proof The proof proceeds by induction on the structure of the subformula of $\neg F^*$. As is common in proofs of this sort, the only delicate argument is that subformulas of the form $\Diamond G^*$ that hold at s in $\mathcal{B}$ also hold at s in $\mathcal{S}$. (Intuitively, this is because $\mathcal{S}$ is a "subballoon" of $\mathcal{B}$.) The argument that subformulas of the form $\Box G^*$ that hold at s in $\mathcal{S}$ also hold at s in $\mathcal{B}$ is exactly dual. All other arguments are trivial.

Within the main induction, we perform an auxiliary induction on the distance from the state considered to $\mathcal{S}$'s bag.

As a base case, we prove the claim for the states in $\mathcal{S}$'s bag. There are several subcases:

- If $\mathcal{S}$'s bag is a singleton $\{s\}$, then s must satisfy $[\mathit{false}]^*$, by construction of σ. Since $\vdash\ [\mathit{false}] \Rightarrow G \equiv \Diamond G$, if $\Diamond G^*$ holds at s in $\mathcal{B}$ then G^* holds at s in $\mathcal{B}$. By induction hypothesis, G^* holds at s in $\mathcal{S}$, and by temporal reasoning $\Diamond G^*$ holds at s in $\mathcal{S}$.

- If $\mathcal{S}$'s bag is a pair $\{s, \hat{s}\}$, then it must be that s is not a halting state, and that R_s holds in no state after s in $\mathcal{B}$'s string (if any) and in no state in $\mathcal{B}$'s bag other than s. Furthermore, $O_s = R_s$. Therefore, s satisfies $R_s \wedge \Box(R_s \Rightarrow [R'_s])^*$ in $\mathcal{B}$, and hence also $\Box R_s$, since $\vdash\ R_s \wedge \Box(R_s \Rightarrow [R'_s]) \Rightarrow \Box R_s$. Assume that s satisfies $\Diamond G^*$ in $\mathcal{B}$. By temporal reasoning, s satisfies $\Diamond(R_s \wedge G^*)$ in $\mathcal{B}$. But s is the last state in $\mathcal{B}$ that satisfies R_s, and so it must be that s satisfies G^*. By induction hypothesis, G^* holds at s in $\mathcal{S}$ as well, and by temporal reasoning $\Diamond G^*$ holds at s in $\mathcal{S}$.

- Otherwise, $\mathcal{S}$'s bag is a subset of $\mathcal{B}$'s bag, and not a singleton. In particular, there is no halting state in the bag. Let R be the disjunction of all O_t for t in $\mathcal{S}$'s bag. By construction of σ, all states in $\mathcal{B}$'s bag that satisfy R are also in $\mathcal{S}$'s bag—the point being that σ makes the biggest cycle possible. In $\mathcal{B}$, it must be that each of the states in $\mathcal{S}$'s bag satisfies $\Box(R \Rightarrow [R'])^*$. This formula simply says that all states in $\mathcal{B}$'s bag that are also in $\mathcal{S}$'s bag must be followed by another state in $\mathcal{S}$'s bag. Moreover, we have that $\vdash R \wedge \Box(R \Rightarrow [R']) \Rightarrow \Box R$. This yields that each state s in $\mathcal{S}$'s bag satisfies $\Box R$ in $\mathcal{B}$'s bag. Let $\Diamond G^*$ hold at s in $\mathcal{B}$'s bag. Then, by temporal reasoning, $\Diamond(R \wedge G^*)$ holds at s in $\mathcal{B}$'s bag. In other words, G^* holds at some state t in $\mathcal{B}$, and t satisfies R. Since t satisfies R, it must also be in $\mathcal{S}$'s bag. Thus, by induction hypothesis, t also satisfies G^* in $\mathcal{S}$, and hence s satisfies $\Diamond G^*$ in $\mathcal{S}$.

Next we consider the states in $\mathcal{S}$'s string. We assume the claim has been proved for all states at distance no bigger than n from $\mathcal{S}$'s bag; we consider the state s at distance $n+1$. Suppose that s satisfies $\Diamond G^*$ in $\mathcal{B}$ and that s is in $\mathcal{S}$'s string. Since $\vdash [R'_s] \wedge \Diamond G \Rightarrow G \vee \Diamond(R_s \wedge \Diamond G)$, either G^* must hold at s, or $\Diamond G^*$ must hold at the state that follows s, in $\mathcal{B}$. In the former case, the complexity of the formula considered has decreased. In the latter case, the complexity of the formula considered has remained the same but we are closer to $\mathcal{S}$'s bag, since s is a state in $\mathcal{S}$'s string and t follows s. In either case, the induction hypothesis immediately yields the desired result.

Since all linearizations of $\mathcal{B}$ satisfy $\neg F^*$, we conclude that all linearizations of $\mathcal{S}$ satisfy $\neg F^*$. ∎

The final step in our proof is constructing an interpretation $(\mathbf{S}, I)$ and checking that $(\mathbf{S}, I, \sigma)$ satisfies $\neg F$.

We take $\mathbf{S}$ to be the set of states in $\mathcal{B}$ that occur in σ.

Each proposition symbol P_i that occurs in F has a value at each state in the balloon. We take $I_\rho(P_i)$ to be the set of states of σ where this value is true. Similarly, if A_i is an action symbol in F, then the proposition symbol $[A_i]^*$ has a value at each state in the balloon (because it occurs in $(X \Rightarrow F)^*$). We take $I_\alpha(A_i)$ to be the set of pairs of states (s, t) such that $[A_i]^*$ is true in s. Two slight oddities should be noticed here. The first one is that we are interpreting an action symbol much as a proposition symbol (the second component of the pair, t, plays no role). The second one is that if $[\mathit{false}]^*$ holds in s then $(s, t) \in I_\alpha(A_i)$, somewhat arbitrarily.

All remaining proposition symbols and action symbols can be interpreted at will, as they do not affect the meaning of F.

In order to show that $(\mathbf{S}, I, \sigma) \models \neg F$, we prove a stronger proposition:

Proposition 4 *If G is a subformula of $\neg F$ and s is a state in σ, then G holds at s in σ if and only if G^* holds at s in $\mathcal{S}$. In particular, σ satisfies $\neg F$.*

Proof The proof is by induction on the structure of G. The only nontrivial case is for formulas of the form $[A]$. In this case, we consider separately the subcases where $[\mathit{false}]^*$ holds at s in $\mathcal{S}$ and where it does not. In both subcases, we use that some of the theorems compiled in X hold at s in $\mathcal{S}$; this is true because the theorems hold at s in $\mathcal{B}$ and because they are free of $\Box$ and $\Diamond$.

If $[\mathit{false}]^*$ holds at s, then $[A]^*$ holds at s for every A of interest, since $\vdash [\mathit{false}] \Rightarrow [A]$. Also, if $[\mathit{false}]^*$ holds at s, the construction of σ yields that σ loops at s, thus $[\mathit{false}]$ holds at s in σ; therefore, $[A]$ holds at s in σ for every A, by definition of the semantics of $[\,]$. Thus, $[A]^*$ holds at s in $\mathcal{S}$, and $[A]$ holds at s in σ.

If $[\mathit{false}]^*$ does not hold at s, then we can use the axioms to decompose A into a boolean combination of proposition symbols, primed proposition symbols, and action symbols. More precisely, consider the following primitive-recursive function d:

$$\begin{array}{rcl}
d([P_i]) & \triangleq & [P_i] \\
d([P_i']) & \triangleq & [P_i'] \\
d([A_i]) & \triangleq & [A_i] \\
d([\mathit{false}]) & \triangleq & \mathit{false} \\
d([\neg A]) & \triangleq & \neg d([A]) \\
d([A \wedge B]) & \triangleq & d([A]) \wedge d([B]) \\
d([(\neg P)']) & \triangleq & d([\neg P']) \\
d([(P \wedge Q)']) & \triangleq & d([P' \wedge Q'])
\end{array}$$

One can derive by induction that $[A]$ and $d([A])$ are provably equivalent for every A:

$$\vdash \neg[\mathit{false}] \Rightarrow [A] \equiv d([A])$$

and this is also valid, of course:

$$\models \neg[\mathit{false}] \Rightarrow [A] \equiv d([A])$$

Therefore, it suffices to consider $[A]$ in the cases where A is *false*, a proposition symbol, a primed proposition symbol, or an action symbol:

- $A = \mathit{false}$: We have that s satisfies $\neg[\mathit{false}]^*$ in $\mathcal{S}$. By construction, σ does not fall into a loop in s (though it may loop between s and $\hat{s}$). Therefore, s satisfies $\neg[\mathit{false}]$ in σ.

- $A = P_i$: Since $\vdash \neg[\mathit{false}] \Rightarrow [P_i] \equiv P_i$, we have that s satisfies $[P_i]^*$ in $\mathcal{S}$ if and only if s satisfies P_i in $\mathcal{S}$. Since $\models \neg[\mathit{false}] \Rightarrow [P_i] \equiv P_i$, similarly, s satisfies $[P_i]$ in σ if and only if s satisfies P_i in σ. Finally, the definition of I_ρ yields that s satisfies P_i in $\mathcal{S}$ exactly when s satisfies P_i in σ.

- $A = P_i'$: If s satisfies $[P_i']^*$ in $\mathcal{S}$, then P_i is one of the conjuncts in the assignment R_s. In the construction of σ, the state immediately after s must satisfy R_s, and hence P_i. Therefore, the semantics yields that s satisfies $[P_i']$ in σ. Conversely, suppose that s does not satisfy $[P_i']^*$ in $\mathcal{S}$; then $\neg P_i$ is one of the conjuncts in the assignment R_s. In the construction of σ, the state immediately after s must satisfy R_s, and hence $\neg P_i$. Therefore, the semantics yields that s satisfies $[(\neg P_i)']$ and not $[P_i']$ in σ.

- $A = A_i$: The definition of I_α is designed to make this case trivial.

We derive that $(\mathbf{S}, I, \sigma) \models \neg F$, as a special case. ▮

This concludes the completeness proof. It follows from the proof that PTLA possesses a finite model property, and hence it is decidable. In fact, it seems likely that the validity problem for PTLA is decidable in polynomial space.

5 Conclusions

We have presented a complete proof system for a propositional temporal logic of actions, PTLA. Lamport has considered extensions of the basic temporal logic of actions, and it seems worthwhile to search for axiomatizations of some of them as well.

The simplest extension consists in adding formulas of the form $[A]_{P_0,\ldots,P_n}$ to the logic; this yields the general temporal logic of actions. Roughly, $[A]_{P_0,\ldots,P_n}$ says that the action A will take place the next time that one of $P_0, \ldots, P_n$ changes value. A further extension consists in adding existential quantification over propositions, for hiding internal state. These extensions make it possible to formulate proofs that a program implements another program, and lead to a simple compositional semantics for concurrent systems. As the logic becomes more powerful, however, it becomes more difficult to choose appropriate proof principles. A complete axiomatization might help in this choice.

Acknowledgements

Leslie Lamport encouraged this work, and helped in describing his logic and in defining PTLA. Rajeev Alur, Cynthia Hibbard, and Jim Saxe suggested improvements in the exposition. Bryan Olivier suggested the current version of Axiom 16, strengthening a previous one which was probably flawed.

References

[BK90] J. C. M. Baeten and J. W. Klop, editors. *CONCUR '90, Theories of Concurrency: Unification and Extension*, volume 458 of *Lecture Notes in Computer Science*, Berlin, 1990. Springer-Verlag.

[Gol87] Robert Goldblatt. *Logics of Time and Computation.* Number 7 in CSLI Lecture Notes. CSLI, Stanford, California, 1987.

[GPSS80] D. Gabbay, A. Pnueli, S. Shelah, and Y. Stavi. On the temporal analysis of fairness. In *Seventh Annual ACM Symposium on Principles of Programming Languages*, pages 163–173. ACM, January 1980.

[HC68] G. E. Hughes and M. J. Cresswell. *An Introduction to Modal Logic.* Methuen Inc., New York, 1968.

[HM85] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, January 1985.

[INM84] INMOS. *Occam Programming Manual.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.

[Lam89] Leslie Lamport. A simple approach to specifying concurrent systems. *Communications of the ACM*, 32(1):32–45, January 1989.

[Lam90] Leslie Lamport. A temporal logic of actions. Research Report 57, Digital Equipment Corporation, Systems Research Center, April 1990.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Symposium on the Foundations of Computer Science.* IEEE, November 1977.

[Pnu81] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.

[Pra76] Vaughan R. Pratt. Semantical considerations on Floyd-Hoare logic. In *17th Symposium on Foundations of Computer Science*, pages 109–121. IEEE, October 1976.

[SC85] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logic. *Journal of the ACM*, 32(3):733–749, July 1985.