Maintenance and Prevention: Formalization and Fixpoint Characterization

Munindar P. Singh

Information Systems Division
Microelectronics and Computer Technology Corporation
3500 W. Balcones Center Drive
Austin, TX 78759-5398
USA

msingh@mcc.com
(512)-338-3431 (voice)
(512)-338-3890 (fax)

Abstract

Maintenance and prevention are important concepts in the science of intelligent systems, yet they are usually ignored in favor of concepts related to achievement. We show, however, that maintenance and prevention are neither trivial nor intractable to formalize or compute and, given their value, merit serious research attention. Maintenance and prevention are duals of each other, but cannot be reduced to other known concepts. We formalize maintenance in a general model of actions and time, giving its model theory and axiomatization. We next introduce techniques from theoretical computer science by giving a fixpoint characterization of maintenance. That characterization yields an efficient algorithm for computing maintenance. Variants of this algorithm apply to achievement-like concepts as well. When applicable, it appears superior to traditional planning approaches.

1 Introduction

The issues of prevention and maintenance of different conditions are of great importance in the design of intelligent systems. Informally, an intelligent system may not only need to achieve different goals, but also to maintain safety and prevent harmful conditions. Indeed, planning systems often deal with maintenance under the guise of *protection*: in achieving conjunctive goals, one conjunct is achieved first and while it is protected, other conjuncts are attempted [Waldinger, 1981]. Further, prevention is crucial to the understanding of deontic concepts such as obligation and corresponding communicative concepts such as prohibitions.

Unfortunately, despite their importance, the concepts of prevention and maintenance have not received significant attention in theoretical AI. We show below that these concepts are neither trivial to define or use nor intractable; accordingly, they merit research attention. We argue the following. One, they cannot be trivially derived from well-understood concepts like achievement. Two, they can be a given a technically perspicuous formalization in terms of a model theory and an axiomatization from which several of their useful properties can be derived. Three, this formalization leads to a fixpoint characterization of them as greatest fixpoints of a fairly simple functional. This indicates that they can be computed as efficiently as other related concepts, such as the achievability of goals. Moreover, we also argue that in many cases, fixpoint characterizations can be used instead of traditional planning to yield more efficient computations.

We present our technical framework briefly in the next section. In section 3, we discuss what maintenance and prevention are not and what they are. Next we define our formal language and model, which contains the ingredients essential for our purposes. In section 5, we give a formal semantics and axiomatization of maintenance. Finally, we derive a fixpoint characterization of maintenance and show how it can be used for computing.

2 The Technical Framework, Briefly

The framework described below captures the notions of *choice* and *control*. It admits concurrent actions of differing durations by different agents. An agent typically can choose between a number of actions. An action may lead to a number of states: depending on the actions of other agents. Time here is not required to be discrete or continuous, but can be either.

Figure 1 has a schematic picture of the formal model. Each point in the picture is a moment. Each moment is associated with a possible state of the world (identified by the atomic propositions that hold there) and the knowledge of different agents (determined by a reflexive and transitive alternativeness relation on moments [Moore, 1984]). A partial order on moments denotes temporal precedence.

Figure 1 is labeled with the actions of two agents. The first agent can constrain the future to some extent by choosing action a or action b. If he does action a, then the world progresses along one of the top two branches out of t_0; if he does action b, then it progresses along one of the bottom two branches. However, the agent cannot control what

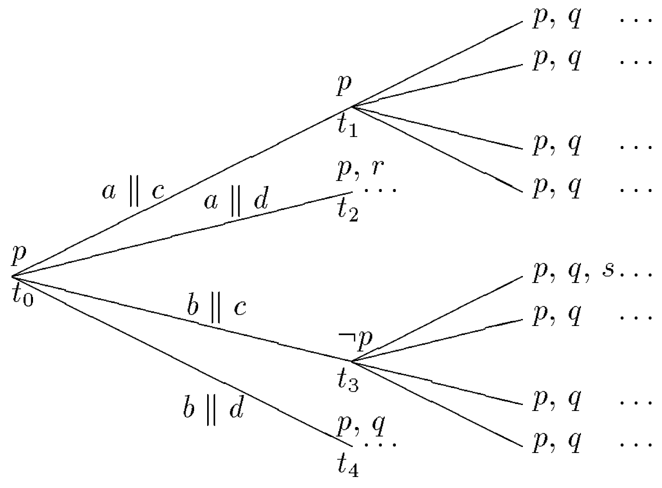

Figure 1: The Formal Model

exactly transpires. For example, if he does action a, then whether t_1 or t_2 becomes the case depends on the actions of the second agent. Both *choice* and *limited control* are thus captured in this model. Actions are package deals. They correspond to the granularity at which an agent can make his choices. In the above example, the first agent can choose between t_1 and t_2, on the one hand, and between t_3 and t_4, on the other hand. However, he can choose neither between t_1 and t_2, nor between t_3 and t_4.

A *scenario* at a moment is a set containing that moment and all future moments along some branch. $p\mathsf{U}q$ is true at a moment t on a scenario, iff q holds at a future moment on the given scenario and p holds on all moments between t and the selected occurrence of q. $\mathsf{F}p$ means that p holds sometimes in the future on the given scenario and abbreviates $\mathsf{trueU}p$. $\mathsf{G}p$ means that p always holds in the future on the given scenario; it abbreviates $\neg\mathsf{F}\neg p$.

The branching-time operator, A, denotes "in *all* scenarios at the present moment." Here "the present moment" refers to the moment at which a given formula is evaluated. A useful abbreviation is E, which denotes "in *some* scenario at the present moment." In other words, $\mathsf{E}p \equiv \neg\mathsf{A}\neg p$. For example, in Figure 1, $\mathsf{EF}r$ and $\mathsf{AF}q$ hold at t_0, since r holds on some moment on some scenario at t_0 and q holds on some moment on each scenario.

We require moments with the same world and knowledge states to have isomorphic future fragments. Such moments can be represented as single moments for representational efficiency. Hence, the partial order of temporal precedence is represented as having cycles: this move saves repetition in computation. Further coherence constraints are imposed to make the models meaningful models of actions. These are essential to the technical results. For reasons of space, we only describe the necessary ones informally as needed.

3 Maintenance and Prevention: Discussion

We focus on the strong sense of maintenance, in which the relevant condition must not only be maintained, but knowingly maintained. Informally, the agent performing the actions to maintain a condition cannot guarantee that the condition is maintained unless he knows that it is.

Intuitively, to maintain a condition means to ensure that it holds forever. And, to prevent a condition means to ensure that it never holds. Thus, it is evident that prevention and maintenance are closely related: a condition can be prevented if and only if its negation can be maintained. For this reason, it is technically viable to restrict ourselves to just one of the two concepts. For reasons of taste (to deal with the positive concept), we consider only maintenance below. We cannot hope to simplify our task further by reducing maintenance to some concept related to achievement, which could then be argued to already be technically known and understood. We now consider some potential such reductions and show that they are fallacious.

Maintenance is not the know-how to achieve $\mathsf{AG}p$. Clearly, if a condition is known to hold forever on all futures, it is trivially maintained. The agent need select no special action to ensure its continued truth. The converse is not true. For example, a condition may be maintainable, but only by repeated actions on part of the agent. In practice, most maintainable conditions fall into the latter category. An extreme example is that of balancing a stick on one's finger; a familiar example is not falling down while standing.

Maintenance is not the know-how to achieve $\mathsf{EG}p$. $\mathsf{EG}p$ means that there is a good scenario, but it does not entail that the agent will be able to prevent a bad scenario: the agent's actions may not be so selective. However, if the agent can maintain p, then under some common assumptions, he should know how to achieve $\mathsf{EG}p$. Indeed, at any moment where he can maintain p, there is at least one scenario where p holds forever.

Maintenance is not the formal dual of know-how to achieve p. Informally, an agent knows how to achieve a condition if he can force it to occur by carefully selecting a finite number of actions. This leads to the following, more plausible suggestion. One might show that maintainability is the formal dual of know-how. But this proposal too falls short.

If an agent does not know how to achieve the negation of the given condition, then he certainly cannot force the given condition to be violated. However, not being able to violate a condition does not entail that the given condition can in fact be maintained. The condition might be one that the agent cannot control one way or the other. That is, for every action of the agent, other agents may perform some actions which variously force the condition to become true or be violated.

3.1 What Maintenance Is

Having discussed what maintenance is not, we are in a better position to formalize what it is. Intuitively, an agent knows how to maintain a condition if he can continually force it to be true, i.e., if he can always perform an action that would counteract the potentially harmful actions of other agents. This entails that not only must the actions of other agents not cause any immediate damage, but the given agent should also ensure that they do not lead to a state where he will not be able to control the situation.

These requirements are not much stronger than the requirements for knowing how to achieve some condition: the key difference is that to achieve something the agent must perform a bounded number of steps, whereas to maintain a condition he may need to keep acting forever. Often, an infinite number of actions would not be necessary, since the agent would need to maintain a condition only long enough for another condition to occur. This, for instance, is the case in planning multiple goals, where the goal achieved first must be protected for only as long as the other goals are being worked on. Also, in finite models, which are of the greatest interest, an agent can maintain p only if he can cycle over some set of safe states.

The precise concept we are trying to formalize is of an agent knowing how to maintain a certain condition p. Therefore, if initially the agent does not know that p holds then he clearly does not know how to maintain it: either it does not hold or he does not believe that it holds. One cannot maintain that which is false and one cannot knowingly maintain that which one believes to be false. This gives us the base case for our definitions.

Suppose that p is known to hold at the given moment. Then, to know how to maintain it, the agent must be able to respond to all eventualities that might cause p to become false. In a traditional game-theoretic setting, this could be accomplished by having the agent choose his action on the basis of the actions of the other agents. However, in our framework, the agents act concurrently. Thus the given agent must choose his action such that no combination of the other agents' actions can violate p. His next action will of course depend on the present actions of others, since the state in which he performs his next action will be determined by the present actions of all. Not only must the agent's chosen action maintain p, it should also maintain his ability to maintain p further. The concept of maintenance differs from achievement in that the agent does not play to win, because there may not be a win state at all, but only to prevent defeat. The agent may be able to force a state in which defeat is impossible, in which case he does not need to select any more actions.

For example, consider Figure 1 again. For simplicity assume that the agent whose actions occur first has perfect knowledge, i.e., his alternativeness relation relates a moment only to itself. Also assume that t_0 is the only successor of all the fringe moments. Then, the agent can maintain p at t_0, since he knows p in t_0 and he can select a in t_0 over which he continues to know p. After that he can just cycle through t_0. He cannot maintain p in t_3, but he can maintain it in all successors of t_3.

4 The Formal Language

Our formal language, $\mathcal{L}$, is based on CTL*, a branching-time logic [Emerson, 1990]; augmentations are described below. $\mathcal{L}$ is the minimal set closed under the following rules. Here $\mathcal{L}_s$ is the set of "scenario-formulae," which is used as an auxiliary definition. The formulae in $\mathcal{L}$ are evaluated relative to moments; those in $\mathcal{L}_s$ are evaluated relative to scenarios and moments. Below, Φ is a set of atomic propositional symbols, $\mathcal{A}$ is a set of agent symbols, $\mathcal{B}$ is a set of basic action symbols, and $\mathcal{X}$ is a set of variables.

L1. $\psi \in \Phi$ implies that $\psi \in \mathcal{L}$

L2. $p, q \in \mathcal{L}$ and $x \in \mathcal{A}$ implies that $p \wedge q$, $\neg p$, $x\mathsf{K}p$, $x\mathsf{M}p$, $(\bigvee a : p) \in \mathcal{L}$

L3. $\mathcal{L} \subseteq \mathcal{L}_s$

L4. $p, q \in \mathcal{L}_s$, $x \in \mathcal{A}$, and $a \in \mathcal{B}$ implies that $p \wedge q$, $\neg p$, $p\mathsf{U}q$, $x[a]p$, $x\langle a\rangle p \in \mathcal{L}_s$

L5. $p \in \mathcal{L}_s$ implies that $\mathsf{A}p \in \mathcal{L}$

4.1 The Formal Model

A model for $\mathcal{L}$ is a tuple, $M = \langle \mathbf{T}, <, \mathbf{A}, [\![\,]\!], \mathbf{B} \rangle$. Here $\mathbf{T}$ is a set of possible moments ordered by $<$. $\mathbf{A}$ assigns agents to different moments; i.e., $\mathbf{A} : \mathbf{T} \mapsto \wp(\mathcal{A})$. $[\![\,]\!]$ is described below. $\mathbf{B}$ assigns alternative moments to the agents at each moment. These are the moments that denote states of affairs that the agents imagine to be the case. $\mathbf{B}$ is used to give the semantics of knowledge (K) and indirectly of maintenance (M).

The relation, $<$, which is a subset of $\mathbf{T} \times \mathbf{T}$, is a strict partial order. Typically, time branches in the future. A scenario at a moment is any single branch of the relation $<$ that begins at the given moment, and contains *all* moments in some linear subrelation of $<$. Different scenarios correspond to different ways in which the world may develop in the future, as a result of the actions of agents and events in the environment. $\mathbf{S}_t$ is the set of all scenarios at moment t. $[S; t, t']$ denotes a period on scenario S from t to t', inclusive. We require $t, t' \in S$ and $t \leq t'$. We label periods with scenarios to allow branching in both the past and the future.

More than one agent may act simultaneously. Basic actions may have different durations relative to one another in different scenarios, including those scenarios that begin at the same moment.

The intension, $[\![\,]\!]$, gives the semantics of atomic propositions and actions. The intension of an atomic proposition is the set of moments at which it is true. The intension of an action symbol a is, for each agent symbol x, the set of periods in the model in which an instance of a is done by x. Thus $t \in [\![p]\!]$ means that p is true at moment t; and, $[S; t, t'] \in [\![a]\!]^x$ means that agent x is performing action a from moment t to moment t'. When $[S; t, t'] \in [\![a]\!]^x$, t' corresponds to the ending of a, but t does not correspond to the initiation of a. This is because a may already be in progress before t. All basic actions take time. That is, if $[S; t, t'] \in [\![a]\!]^x$, then $t < t'$. The superscript denoting the agent is elided when it can be understood from the context.

4.2 Semantics

The semantics of sentences, i.e., formulae, in the formal language is given relative to a model and a moment in it. $M \models_t p$ expresses "M satisfies p at t." This is the main notion of satisfaction. For formulae in $\mathcal{L}_s$, $M \models_{S,t} p$ expresses "M satisfies p at moment t on scenario S" (we require that $t \in S$). We say p is *satisfiable* iff for some M and t, $M \models_t p$; we say p is *valid* in M iff it is satisfied at all moments in M. It is assumed that each action symbol is quantified over at most once in any formula. Below, $p|^a_b$ is the formula resulting from the substitution of all occurrences of a in p by b. We define $\mathsf{false} \equiv (p \wedge \neg p)$, for any $p \in \Phi$, and $\mathsf{true} \equiv \neg\mathsf{false}$. Formally, we have:

M1. $M \models_t \psi$ iff $t \in [\![\psi]\!]$, where $\psi \in \Phi$

M2. $M \models_t p \wedge q$ iff $M \models_t p$ and $M \models_t q$

M3. $M \models_t \neg p$ iff $M \not\models_t p$

M4. $M \models_t \mathsf{A}p$ iff $(\forall S : S \in \mathbf{S}_t \Rightarrow M \models_{S,t} p)$

M5. $M \models_t (\bigvee a : p)$ iff $(\exists b : b \in \mathcal{B}$ and $M \models_t p|^a_b)$, where $p \in \mathcal{L}$

M6. $M \models_{S,t} p\mathsf{U}q$ iff $(\exists t' : t \leq t'$ and $M \models_{S,t'} q$ and $(\forall t'' : t \leq t'' \leq t' \Rightarrow M \models_{S,t''} p))$

M7. $M \models_{S,t} x[a]p$ iff $(\forall t' \in S : [S; t, t'] \in [\![a]\!]^x$ implies that $(\exists t'' : t < t'' \leq t'$ and $M \models_{S,t''} p))$

M8. $M \models_{S,t} x\langle a\rangle p$ iff $(\exists t' \in S : [S; t, t'] \in [\![a]\!]^x$ and $(\exists t'' : t < t'' \leq t'$ and $M \models_{S,t''} p))$

M9. $M \models_{S,t} p \wedge q$ iff $M \models_{S,t} p$ and $M \models_{S,t} q$

M10. $M \models_{S,t} \neg p$ iff $M \not\models_{S,t} p$

M11. $M \models_{S,t} p$ iff $M \models_t p$, where $p \in \mathcal{L}$

M12. $M \models_t x\mathsf{K}p$ iff $(\forall t' : (t, t') \in \mathbf{B}(x)$ implies $M \models_{t'} p)$

4.3 Action Operators: Discussion

$\mathcal{L}$ also contains operators on actions. For an action symbol a, an agent symbol x, and a formula p, $x[a]p$ holds on a given scenario S and a moment t on it, iff, if x performs a on S starting at t, then p holds at some moment while a is being performed. The formula $x\langle a\rangle p$ holds on a given scenario S and a moment t on it, iff, x performs a on S starting at t and p holds at some moment while a is being performed. These definitions require p to hold at any moment in the (left-open and right-closed) period in which the given action is being performed. Thus they are weaker than possible definitions that require p to hold at the moment at which the given action completes.

These definitions require p to hold at any moment in the (left-open and right-closed) period in which the given action is being performed. Thus they are weaker than possible definitions that require p to hold at the moment at which the given action completes. This is a necessary generalization since we allow moments to occur between the starting and ending of an action.

$[\,]$ and $\langle\,\rangle$ yield scenario-formulae, which can be combined with the operators A and E. Thus $\mathsf{A}[a]p$ denotes that on all scenarios S at the present moment, if a is performed on S, then p holds at some moment on S between the present moment and the moment at which a is completed. Similarly, $\mathsf{E}\langle a\rangle p$ denotes that a is being done on some scenario at the present moment and that on this scenario p holds at some moment between the present moment and the moment at which a is completed.

Existential quantification over basic actions is a useful feature. Of the several basic actions that an agent may do at a given moment, we often like to restrictively talk of the subset of actions that have some interesting property. Indeed, we need something like this to formally express the idea of *choice:* an agent may be able to do several actions, but would, in fact, choose to do one. For each action that an agent may choose to do, there is a set of scenarios over which those actions are attempted and done.

5 Formalization of Maintenance

We formalize maintenance using the auxiliary notion of a tree of actions. A tree can be (a) empty, (b) a single action, or (c) a single action (its *radix*) adjoined with a set of subtrees (each of which has a different radix). Trees are used to encode the choices made by an agent in maintaining p. The radix represents the current choice. The depth of a tree is the number of actions along any branch of it. We require all subtrees of a tree to be of equal depth.

Let Υ be the set of trees. Then, (a) $\emptyset \in \Upsilon$, where $\emptyset$ is the empty tree; (b) $a \in \Upsilon$, where $a \in \mathcal{B}$; and (c) $\langle a; \tau_1, \ldots, \tau_m\rangle \in \Upsilon$, where $a \in \mathcal{B}$ and $\tau_1, \ldots, \tau_m \in \Upsilon$ are m trees with different radices but the same depth.

An agent maintains p over an empty tree if he knows that p holds currently. He maintains p over a single action if he knows that he scan perform that action in the given state and p holds over the duration of the action and in its terminal state. He maintains p over a general tree if he maintains it over its initial action and then over some applicable subtree. Define $[\![\tau]\!]_{t,p}$ as the set of periods beginning at t over which p is maintained by tree τ. These are the periods over which the agent can knowingly select the right actions.

Since we are not limited to discrete models with unitlength actions, $\langle\,\rangle$ and $[\,]$ are not duals of each other. $\neg[a]\neg p$ means that a is performed and p holds at every moment during a. Thus, we have the following definitions.

- $[\![\emptyset]\!]_{t,p} = (\text{if } M \models_t \mathsf{K}p, \text{ then } \{[S; t, t]\} \text{ else } \{\,\})$
- $[\![a]\!]_{t,p} = \{[S; t, t'] \in [\![a]\!] : M \models_t \mathsf{K}(\mathsf{A}\neg[a]\neg\mathsf{K}p \wedge \mathsf{E}\langle a\rangle\mathsf{true})\}$

  In other words, the agent maintains p over $[S; t, t']$ iff the agent knows at t that he will know p holds along all outcomes of performing a.

- $[\![\langle a; \tau_1, \ldots, \tau_m\rangle]\!]_{t,p} = \{[S; t, t''] : (\forall t_k : (t, t_k) \in \mathbf{B} \Rightarrow (\exists S_k, t'_k : [S_k; t_k, t'_k] \in [\![a]\!]_{t_k,p}$ and $(\forall S_k, t'_k : [S_k; t_k, t'_k] \in [\![a]\!]_{t_k,p} \Rightarrow (\exists t''_k, j : [S_k; t'_k, t''_k] \in [\![\tau_j]\!]_{t'_k,p})\}$

These definitions, though complex, yield a surprisingly simple axiomatization. An agent knows how to maintain p to depth i if there is a tree of depth i over which he maintains p. An agent knows how to maintain p if he can maintain it to all depths.

M13. $M \models_t \mathsf{M}^i p$ iff $(\exists \tau : \text{depth}(\tau) = i \text{ and } [\![\tau]\!]_{t,p} \neq \{\,\})$

M14. $M \models_t \mathsf{M}p$ iff $(\forall i : \mathsf{M}^i p)$

By constructing trees out of smaller trees, we can show the following.

Lemma 1 $(i \geq j) \Rightarrow (\mathsf{M}^i p \wedge \mathsf{M}^j p) \equiv \mathsf{M}^i p$ □

Lemma 2 $\mathsf{M}^i \mathsf{M}^i p \equiv \mathsf{M}^{2i} p$ □

5.1 Axiomatization of Maintenance

Km1. $\mathsf{K}p \wedge (\bigvee a : \mathsf{K}(\mathsf{E}\langle a \rangle \mathsf{true} \wedge \mathsf{A}\neg[a]\neg \mathsf{M}p)) \rightarrow \mathsf{M}p$

Km2. All substitution instances of the validities of the underlying logic.

Theorem 3 Axioms Km1 and Km2 are sound and complete for M.

The proof is through a straightforward canonical model construction and is not included for reasons of space. □

Lemma 4 $\mathsf{KAG}p \rightarrow \mathsf{M}p$ □

Observation 5 $\mathsf{AG}(p \rightarrow q) \Rightarrow (\mathsf{M}p \rightarrow \mathsf{M}q)$ □

Observation 6 $\mathsf{M}p \rightarrow p$ □

Observation 7 $\mathsf{MM}p \rightarrow \mathsf{M}p$ □

6 Computing Maintenance

Computationally, one may use a formal semantics and axiomatization of a concept in various ways, including automatic or guided theorem provers, and tools for checking prespecified models. The latter approach has proved particularly effective in program verification with temporal logic specifications. We adapt it for reasoning about action. We assume that we are given a model and a formula $\mathsf{M}p$ and attempt to compute all moments at which $\mathsf{M}p$ holds in it.

6.1 The Propositional Mu-Calculus

An especially powerful class of tools (e.g., [Burch *et al.*, 1990]) is based on the mu-calculus of [Scott & de Bakker, 1969]. The mu-calculus provides a syntax for writing expressions involving least and greatest fixpoints. In order to use mu-calculus based algorithms, we show how maintenance can be characterized as a greatest fixpoint of a moderately simple functional. Indeed, this expression is obvious from the axiomatization given in section 5.1.

Our presentation below is self-contained though simplified to just capture the features we need. Let $\mathcal{L}^\mu$ be $\mathcal{L}$ extended to accommodate the mu-calculus. Let Ξ be a set of propositional variables. The operator ν binds variables. $\zeta(Z)$ denotes a functional on $Z \in \Xi$, i.e., an expression free in Z in which Z occurs inside an even number of negations. This entails that $\zeta(Z)$ is monotone in Z and, by the Tarski-Knaster theorem, has both a greatest and a least fixpoint.

L6. Replace $\mathcal{L}$ with $\mathcal{L}^\mu$ and $\mathcal{L}_s$ with $\mathcal{L}_s^\mu$ in the syntax rules of section 4.

L7. $Z \in \Xi$ implies that $Z \in \mathcal{L}^\mu$

L8. $Z \in \Xi$, $\zeta(Z) \in \mathcal{L}^\mu$ implies that $(\nu Z : \zeta(Z)) \in \mathcal{L}^\mu$

For $p \in \mathcal{L}$, the semantics of section 4.2 holds, but is rephrased to assign to p the set of moments where it holds. The functionals in $\mathcal{L}^\mu$ are semantically functions from sets of moments to sets of moments. $(\nu Z : \zeta(Z))$ is the greatest fixpoint of the functional $\zeta(Z)$. By the Tarski-Knaster theorem, $(\nu Z : \zeta(Z)) = \bigcap_{i \geq 0} \zeta^i(\mathsf{true})$, where ζ^i denotes ζ composed i times.

M15. $t \in [\![p]\!]$ iff $M \models_t p$, where $p \in \mathcal{L}$

M16. $S \in [\![p]\!]_S$ iff $M \models_{S,t} p$, where $p \in \mathcal{L}_s$ and $S \in \mathbf{S}_t$

M17. $[\![\zeta]\!]$ is a function $\mathbf{T} \mapsto \mathbf{T}$ defined in the obvious manner.

M18. $[\![(\nu Z : \zeta(Z))]\!] = \bigcap_{i \geq 0} \zeta^i(\mathsf{true})$

Theorem 8 $\mathsf{M}p \equiv (\nu Z : \mathsf{K}p \wedge (\bigvee a : \mathsf{K}(\mathsf{E}\langle a \rangle \mathsf{true} \wedge \mathsf{A}\neg[a]\neg Z)))$ □

An algorithm for computing the above formula is given next. It reduces the computation of $\mathsf{M}p$ to a reachability analysis, albeit of a slightly sophisticated variety. An obvious observation is that in calculating $(\nu Z : \zeta(Z)) = \bigcap_i \zeta^i(\mathsf{true})$, we can limit i to be the first j such that $\zeta^j(\mathsf{true}) = \zeta^{j+1}(\mathsf{true})$. Since $[\![\mathsf{true}]\!] = \mathbf{T}$ and at least one moment must be removed by each iteration of ζ, we have that $j \leq |\mathbf{T}|$. However, even a naive algorithm would not execute more than d iterations, where d is the path in $\mathbf{T}$ with the greatest number of actions. We assume that there be a finite number of actions between any pair of connected moments. Thus termination is guaranteed if $\mathbf{T}$ is finite or if all paths in it loop back to a prior moment.

```
procedure eval (f, binding-list)
  ifatomic-proposition?(f)
    base-eval(f)
  if variable?(f)
    get-binding(f,binding-list)
  case f
    $(\nu Z : \zeta(Z))$:res := $\mathsf{true}$; next := $\zeta(\mathsf{true})$
                  while res $\neq$ next do
                        res := next; new-b := ((Z next) binding-list)
                        next := eval($\zeta$(next), new-b); endwhile
    else: compute as in the semantic definitions
```

We can improve this algorithm further by using Lemma 2, which shows that to compute $\mathsf{M}^{2i}p$, we need just one more iteration than for $\mathsf{M}^i p$. In other words, we require only $\log d$

iterations, where d is as above. To do this requires, in effect, that we treat trees of actions as (abstract) actions, i.e., to construct abstract actions on the fly. This happens naturally in our approach, since we can give a semantics to $\langle\,\rangle$ and $[\,]$ for trees based on $[\![\,]\!]_{t,p}$, which we defined above. Hence, we have the following (knowledge is rolled into the definition of $[\![\,]\!]_{t,p}$).

Theorem 9 $\mathsf{M}p \equiv (\nu Z : \mathsf{K}p \wedge (\bigvee \tau : \mathsf{A}\neg[\tau]\neg Z))$ $\Box$

Given trees of depth i, trees of depth $2i$ can be constructed directly. Thus we obtain behavior akin to an iterative squaring algorithm.

7 Conclusions

We showed how maintenance, which potentially requires unbounded actions, can be formalized and given a simple axiomatization. We also showed how it can be given a fixpoint characterization leading to a simple algorithm for computing it. Such algorithms can be fruitfully developed for other modalities pertaining to reasoning about action, e.g., by allowing least fixpoints and combinations of least and greatest fixpoints.

References

[Burch *et al.*, 1990] Burch, J. R.; Clarke, E. C.; McMillan, K. L.; Dill, D. L.; and Hwang, L. J.; 1990. Symbolic model checking: 10^{20} states and beyond. In *LICS*.

[Emerson, 1990] Emerson, E. A.; 1990. Temporal and modal logic. In Leeuwen, J.van, editor, *Handbook of Theoretical Computer Science*, volume B. North-Holland Publishing Company, Amsterdam, The Netherlands.

[Moore, 1984] Moore, Robert C.; 1984. A formal theory of knowledge and action. In Hobbs, Jerry R. and Moore, Robert C., editors, *Formal Theories of the Commonsense World.* Ablex Publishing Company, Norwood, NJ. 319–358.

[Scott & de Bakker, 1969] Scott, Dana and de Bakker, J.; 1969. A theory of programs. Technical report, IBM, Vienna.

[Waldinger, 1981] Waldinger, Richard; 1981. Achieving several goals simultaneously. In Webber, Bonnie L. and Nilsson, Nils J., editors, *Readings in Artificial Intelligence.* Morgan Kaufmann. 250–271.