Applying the Mu-Calculus in Planning and Reasoning about Action

Munindar P. Singh
Department of Computer Science
Box 7534
North Carolina State University
Raleigh, NC 27695-7534, USA

singh@ncsu.edu

Abstract

Planning algorithms have traditionally been geared toward achievement goals in single-agent environments. Such algorithms essentially produce plans to reach one of a specified set of states. More general approaches for planning based on temporal logic (TL) are emerging. Current approaches tend to use linear TL, and can handle sets of sequences of states. However, they assume deterministic actions with all changes effected solely by one agent.

By contrast, we use a branching model of time that can express concurrent actions by multiple agents and the environment, leading to nondeterministic effects of an agent's actions. For this reason, we view plans not as sequences of actions, but as decision graphs describing the agent's actions in different situations. Thus, although we consider single-agent decision graphs, our approach is better suited to multiagent systems. We also consider an expressive formalism, which allows a wider variety of goals, including achievement and maintenance goals. Achievement corresponds to traditional planning, but maintenance is more powerful than traditional maintenance goals, and may require nonterminating plans.

To formalize decision graphs requires a means to "alternate" the agent's and the environment's choices. From logics of program, we introduce the propositional *mu-calculus*, which has operators for least and greatest fixpoints. We give a semantics, a fixpoint characterization, and an algorithm to compute decision graphs.

1 Introduction

There is an increasing interest in systems based on intelligent agents, which are expanding into a variety of important mainstream applications. Consequently, such agent-based intelligent systems must be built and verified to the same rigorous standards we expect elsewhere. This speaks to the importance of logic-based approaches for agent design, and highlights the potential usefulness of classical logics of program and other formal techniques. There is some historical support for applying conventional techniques in AI. For instance, dynamic logic was introduced into the AI and logics of program communities at about the same time. The applicability of temporal

logic techniques in planning is well recognized and is studied by several researchers, for example, Bacchus & Kabanza [1].

Planning and reasoning about action are widely regarded as among the key components of intelligent agency, and are ideal targets for the application of formal techniques. This paper studies these subjects in an abstract setting. We characterize planning problems in a way that better captures their structure than in traditional approaches, and show how conventional logics of program techniques can be adapted in solving them. In particular, we introduce *control progression* as a planning technique that selects actions with which the agent can control a branching future.

Motivation. Roughly, planning involves finding a series of choices through which an agent can accomplish a given task. Traditionally, planners have considered tasks that correspond to simple goals of achievement, which are satisfied when a desired state is reached. However, it is now recognized that *temporally extended* goals are often necessary, for example, see Bacchus & Kabanza [1]. These goals are satisfied when a desired sequence of states is achieved.

Although achievement goals are usually considered, we can also allow maintenance goals. Much as one would expect, achievement goals call for the attainment of states in which the given condition is satisfied, whereas maintenance goals call for continual action to preserve the truth of the given condition. Maintenance goals are important, because an agent may not only need to achieve different states, but also to maintain safety conditions and, equivalently, to prevent harmful conditions. This basic idea is common in theories of know-how (to achieve a condition), for example, see Singh [26] and seeing-to-it-that, for example, see Belnap & Perloff [3] and Chellas [7]. However, existing theories tend to focus exclusively on achieving a condition, rather than maintaining one, or achieving and then maintaining a condition.

The idea of maintenance also emerges in implemented planning systems, which from the time of Waldinger [30] have considered maintenance as a key functionality in constructing effective plans, especially when more than one goal must be brought about. Maintenance features also in the dMARS system [15]. It is therefore quite interesting that maintenance has not received corresponding attention in the formal reasoning about action community. Consequently, implemented systems usually handle maintenance in a seat-of-the-pants manner.

Previous TL approaches assume that all changes in the environment are caused by the agent through deterministic actions. We propose a relaxed approach that allows concurrent actions by different agents. This leads naturally to a model of time that allows multiple future *paths* from each moment [11]. Plans in such a model cannot be sequences of actions, but must be decision trees or graphs that describe the agent's actions under different situations.

Branching TLs include *path quantifiers*, which allow us to make assertions about all or some of the paths. However, branching TLs are not suited to capturing decision trees as described above. This is because universal path quantification is too strong, and existential path quantification is too weak. Typically, a good action will not lead to success on all paths that can result from it, and a bad action may lead to success on some path that can result from it. What we require is a systematic means to obtain the effect of an alternation of the agent's choices with moves by the environment (although these may be physically concurrent, of course). This logical

alternation cannot be readily captured in TLs, because different parties have to choose alternating actions. For example, if a goal like $\mathsf{AGEF}p$ (in the notation of CTL [11], and formally defined below) is considered, then it is not clear what is being planned by the agent and what arises due to environmental effects. Recall that $\mathsf{AGEF}p$ is true in a moment where at all future moments on all branches, there will always be a branch on which there will be a moment where p holds. As a result, $\mathsf{AGEF}p$ turns out to be stronger than maintenance, because a state as described by $\mathsf{AGEF}p$ may not be found even though p can be maintained.

Approach. However, this effect is obtained by specifying the desired behavior in terms of the *propositional mu-calculus*, or the *mu-calculus*, for short [16]. The mu-calculus includes operators to specify the greatest and least fixpoints of expressions. It leads to a succinct formulation of achievement and maintenance goals. Algorithms to compute mu-calculus expressions exist, and can be adapted for planning.

The mu-calculus is a generalization of temporal and dynamic logics. Our contribution is in showing how it can be applied in formulating planning problems, and developing algorithms for them. Our approach applies to achievement goals as well as maintenance goals, but we emphasize the latter, because they are more interesting. A benefit of our approach is that the nature of the goal is itself expressed in the object language. Thus goals can be nested, for example, "achieve a state where the agent can maintain a condition."

We choose a formal language related to CTL* and dynamic logic [11] as popularized in the AI community by, among others, Rao & Georgeff [21] and Singh [26]. Another option would be the situation calculus, whose recent versions even allow concurrency, nondeterminism, and even continuity, for example, see Miller [18] and Reiter [23]. However, we would need additional extensions to these approaches to allow path quantifiers and operators for fixpoints as in our approach.

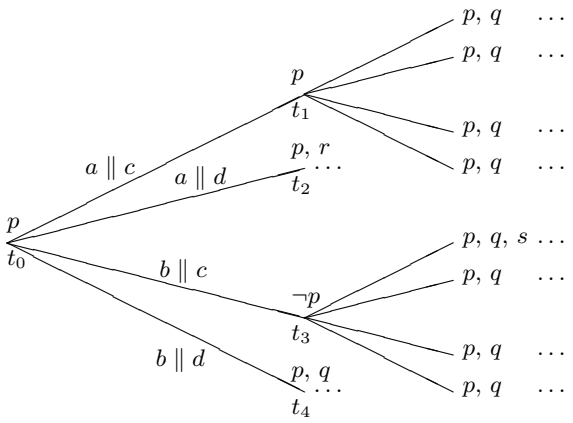
Although fixpoint calculations are commonly used implicitly in the interpreters of the logic programming languages, explicit fixpoint calculations as developed here are less common there.

Organization. Section 2 describes our formal language and model. Section 3 discusses achievement and maintenance conceptually. Section 4 gives a formal semantics maintenance along with a recursive characterization. Finally, Section 5 derives a fixpoint characterization of maintenance and outlines an algorithm to compute maintenance plans. Throughout, we relate our approach to one for achievement.

2 The Technical Framework

The framework described below combines time and nondeterministic actions; it is related to the frameworks of Chellas [7], Rao & Georgeff [21], and Singh [26]. We consider discrete time with concurrent, unit-length atomic actions. As a consequence, there is a finite number of actions between any pair of connected moments.

Figure 1 shows the formal model. Each point in the picture is a *moment*. Each moment defines a unique possible *state* of the world. The term *state* is used informally in the literature. For our technical purposes, the state that holds at a moment is given



(x is the agent whose actions are listed first)

Figure 1: The Formal Model

by the atomic propositions that hold at that moment. A state may occur at several moments. Each moment also identifies the knowledge each agent has in that moment. A partial order on moments denotes temporal precedence.

Figure 1 is labeled with the actions of two agents. The first agent, x, 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 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. Consequently, in Figure 1, x can choose between the set $\{t_1, t_2\}$ and the set $\{t_3, t_4\}$. However, he cannot choose among the members of the above sets.

The state of the world that holds at a moment is identified by the atomic propositions that hold there. We require moments with the same world states to have isomorphic future fragments. This corresponds to the constraint of *weak determinism*, motivated in [26]. It means that although the individual actions are nondeterministic, the set of possible actions and outcomes depends on the given state. If, in addition, two moments have the same knowledge for all the agents, we can treat them as a single moment for representational efficiency. Hence, the partial order of temporal precedence may be represented as having cycles: this representation saves repetition in computation.

2.1 Syntax

$\mathcal{ACTL}^*$ ("A" for actions), our formal language, enhances CTL*, a branching-time logic [11]. $\mathcal{ACTL}^*{}_\pi$ is an auxiliary definition and includes the "path-formulae." Below, Φ is a set of atomic propositional symbols and $\mathcal{X}$ is a set of agent symbols. $\mathcal{A}$ is a set of basic action symbols, partitioned into $\mathcal{A}_c$ and $\mathcal{A}_v$, sets of constant and variable action symbols, respectively. Informally, $x\mathsf{K}p$, $x\mathsf{K}_\mathsf{a}p$, and $x\mathsf{K}_\mathsf{m}p$ respectively mean that x knows that p, x knows how to *a*chieve p, and x knows how to *m*aintain p.

L1. $\psi \in \Phi$ implies that $\psi \in \mathcal{ACTL}^*$

L2. $p, q \in \mathcal{ACTL}^*$, $x \in \mathcal{X}$, and $\alpha \in \mathcal{A}_v$ implies that $p \wedge q$, $\neg p$, $x\mathsf{K}p$, $x\mathsf{K}_\mathsf{m}p$, $x\mathsf{K}_\mathsf{a}p$, $(\bigvee \alpha : p) \in \mathcal{ACTL}^*$

L3. $\mathcal{ACTL}^* \subseteq \mathcal{ACTL}^*{}_\pi$

L4. $p, q \in \mathcal{ACTL}^*{}_\pi$, $x \in \mathcal{X}$, and $a \in \mathcal{A}$ implies that $p \wedge q$, $\neg p$, $p\mathsf{U}q$, $x[a]p \in \mathcal{ACTL}^*{}_\pi$

L5. $p \in \mathcal{ACTL}^*{}_\pi$ implies that $\mathsf{A}p \in \mathcal{ACTL}^*$

For notational simplicity, we use the following naming conventions in this paper.

- x, etc. are agents
- ψ, etc. are atomic propositions
- p, q, r, etc. are formulae in $\mathcal{ACTL}^*$ or $\mathcal{ACTL}^*{}_\pi$ (determined by the surrounding syntax—this is essential because of rule L3)
- a, b, etc. are basic actions, constant or variable
- α, etc. are variables that range over basic actions
- t, etc. are moments
- P, etc. are paths.

2.2 Formal Model

A model for $\mathcal{ACTL}^*$ is a tuple, $M = \langle \mathbf{T}, <, \mathbf{A}, [\![\,]\!], \mathbf{K} \rangle$, where we have the following.

- $\mathbf{T}$ is a set of possible moments.
- $\mathbf{A}$ assigns agents to different moments; that is, $\mathbf{A} : \mathbf{T} \mapsto \wp(\mathcal{X})$.
- The relation $< \subseteq \mathbf{T} \times \mathbf{T}$ is a discrete and finitely branching partial order on $\mathbf{T}$. $\mathbf{P}_t$ is the set of all paths induced by $<$ that begin at moment t. We assume that for all t, $\mathbf{P}_t \neq \emptyset$. $[P; t, t']$ denotes a period on path P from t to t', inclusive—we require $t, t' \in P$ and $t \leq t'$. We label periods with paths to allow branching in both the past and the future. Formally, $[P; t, t']$ is the intersection of P with the set of moments between t and t', both inclusive. Thus it is possible that $[P; t, t'] = [P'; t, t']$ even though $P \neq P'$.
- 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 constant 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, $[P; t, t'] \in [\![a]\!]^x$ means that agent x is performing action a from moment t to moment t'. When $[P; t, t'] \in [\![a]\!]^x$, t corresponds to the initiation of a and t' to its ending. Basic actions take time. That is, if $[P; t, t'] \in [\![a]\!]^x$, then $t < t'$.
- $\mathbf{K}$ assigns to each agent a binary relation on moments. $(t, t') \in \mathbf{K}(x)$ means that for agent x moments t' is an epistemic alternative for moment t. We assume that, for each agent x, $\mathbf{K}(x)$ is reflexive and transitive.

A useful notation is the $[\![\,]\!]^x_t$, which indicates the projection of $[\![\,]\!]^x$ on to moment t.

Definition 1 $[\![a]\!]^x_t \stackrel{\text{def}}{=} \{[P;t,t'] : [P;t,t'] \in [\![a]\!]^x\}$

Perfect knowledge means that the agent knows every proposition that is true. This is captured by making $\mathbf{K}(x)$ the self-loop relation over $\mathbf{T}$. Our approach does not assume perfect knowledge, although some of our examples do so, for simplicity.

Definition 2 $\mathbf{K}$ denotes *perfect knowledge* of agent x when $\mathbf{K}(x) = \{(t,t) : t \in \mathbf{T}\}$

2.3 Semantics

The semantics of $\mathcal{ACTL}^*$ 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 ${\mathcal{ACTL}^*}_\pi$, $M \models_{P,t} p$ expresses "M satisfies p at moment t on path P" (we require that $t \in P$). This is an auxiliary notion of satisfaction. 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. Each action symbol is quantified over at most once in any formula. Below, $p|^\alpha_b$ is the formula resulting from the substitution of all occurrences of α 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 $(\forall P : P \in \mathbf{P}_t \Rightarrow M \models_{P,t} p)$

M5. $M \models_t (\bigvee \alpha : p)$ iff $(\exists b : b \in \mathcal{A}_c$ and $M \models_t p|^\alpha_b)$

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

M7. $M \models_{P,t} x[a]p$ iff $[P;t,t'] \in [\![a]\!]^x$ implies $M \models_{P,t'} p$

M8. $M \models_{P,t} p \wedge q$ iff $M \models_{P,t} p$ and $M \models_{P,t} q$

M9. $M \models_{P,t} \neg p$ iff $M \not\models_{P,t} p$

M10. $M \models_{P,t} p$ iff $M \models_t p$, where $p \in \mathcal{ACTL}^*$

M11. $M \models_t x\mathsf{K}p$ iff $(\forall t' : (t,t') \in \mathbf{K}(x)$ implies $M \models_{t'} p)$

The semantic definitions of $\mathsf{K_a}$ and $\mathsf{K_m}$ are not included above. They are discussed at length and given below.

We define t and t' to be congruent moments as follows. As explained above, congruent moments can be represented as a single state.

Definition 3 $t \cong t' \stackrel{\text{def}}{=} (\forall \psi \in \Phi : M \models_t \psi$ iff $M \models_{t'} \psi)$ and $(\forall x, t'' : (t,t'') \in \mathbf{K}(x)$ iff $(t',t'') \in \mathbf{K}(x))$

2.4 Temporal and Action Operators

$p\mathsf{U}q$ is true at a moment t on a path iff q holds at a future moment on the given path P, and p holds on all moments between t and the selected occurrence of q. F and G are abbreviations. $\mathsf{F}p$ means that p holds sometimes in the future on P. $\mathsf{G}p$ means that p always holds in the future on P. The branching-time operator, A, denotes "in *all* paths at the present moment." Here "the present moment" refers to the moment at which a given formula is evaluated. E is an abbreviation denoting "in *some* path at the present moment."

Definition 4 $\mathsf{F}p \stackrel{\text{def}}{=} \mathsf{true}\mathsf{U}p$.

Definition 5 $\mathsf{G}p \stackrel{\text{def}}{=} \neg\mathsf{F}\neg p$.

Definition 6 $\mathsf{E}p \stackrel{\text{def}}{=} \neg\mathsf{A}\neg p$.

Example 7 In Figure 1, $\mathsf{EF}r$ and $\mathsf{AF}q$ hold at t_0, since r holds on some moment on some path at t_0 and q holds on some moment on each path. Similarly, $\mathsf{AG}(p \vee \neg p)$ holds at t_0 because $(p \vee \neg p)$ holds at every moment in the future of t_0. Assuming p always holds on the top path in the picture, $\mathsf{EG}p$ also holds at t_0.

For an action symbol a, an agent symbol x, and a formula p, $x[a]p$ holds on a given path P and a moment t on it iff if x performs a on P starting at t, then p holds at the moment where a completes. $x\langle a\rangle p$ is the dual of $x[a]p$.

Definition 8 $x\neg\langle a\rangle\neg p \stackrel{\text{def}}{=} x[a]p$

$\mathsf{A}[a]p$ denotes that on all paths P at the present moment, if a is performed on P, then p holds when that execution of a is completed. Similarly, $\mathsf{E}\langle a\rangle p$ denotes that a is done on some path at the present moment and that p holds when that execution of a is completed.

Example 9 Figure 1 satisfies the following at t_0:

(a) $\mathsf{E}x\langle a\rangle\mathsf{true}$

(b) $\mathsf{E}x\langle b\rangle p \wedge \mathsf{E}x\langle b\rangle\neg p$

(c) $\mathsf{A}x[a]p$

(d) $\mathsf{A}x[d]\mathsf{false}$ (since x does not perform d at t_0).

Existential quantification over basic actions enables us to restrictively talk of interesting sets of actions. $(\bigvee \alpha : p)$ means that substituting some action symbol for occurrences of α in p yields a true formula. We define $\bigwedge$ as the dual of $\bigvee$.

Definition 10 $(\bigwedge \alpha : p) \stackrel{\text{def}}{=} \neg(\bigvee \alpha : \neg p)$

Example 11 Figure 1 satisfies the following at t_0:

(a) $(\bigvee \alpha : \mathsf{E}x\langle\alpha\rangle\neg p)$ (since b can be substituted for α)

(b) $(\bigvee \alpha : \mathsf{A}x[\alpha]p)$ (since a can be substituted for α).

Because formulae of the form $(\bigvee \alpha : x\mathsf{K}(\mathsf{E}\langle\alpha\rangle\mathsf{true} \wedge \mathsf{A}[\alpha]p))$ arise frequently in our discussion, we define an abbreviation C, which we term the *control necessitation operator*. $\mathsf{C}p$ means that the agent knows an action α, which he can perform, and of which he knows that on all paths where he performs α, p comes to hold when it ends.

Definition 12 $\mathsf{C}p \stackrel{\text{def}}{=} (\bigvee \alpha : \mathsf{K}(\mathsf{E}\langle\alpha\rangle\mathsf{true} \wedge \mathsf{A}[\alpha]p))$

3 Achievement and Maintenance, Conceptually

We consider only a single agent in the remainder of this paper. Other agents and the environment can be thought of as being implicit in the nondeterminism of the given agent's actions.

Broadly put, there are two main classes of approaches to know-how, discussed at length in [27]. One class follows traditional philosophical intuitions in separating ability from opportunity, for example, Brown [6] and van der Hoek *et al.* [29]. This class of approaches preserves the natural language meaning of knowing how to do something even if one cannot actually do it. However, this naturalness comes at the price of defining know-how based on counterfactual situations. In contrast, the second class is situated and considers the ability as manifest in the opportunities of the given situation. Here, the ability and opportunity go hand-in-hand, and an agent cannot have one without the other. This class is exemplified by Belnap & Perloff [3], Chellas [7], and Singh [26]. Our present approach is in the latter category. This facilitates reading the figures and examples below: what you see is what you get!

3.1 Achievement

Intuitively, an agent knows how to achieve a condition if he can knowingly force it to become true. In other words, the agent can select an action after which the given condition is obtained or a state is achieved from where the agent can perform further necessary actions. The key idea is that along any path where the agent exercises his know-how only a finite number of actions should need to be performed to obtain the given condition. If the condition is already known to hold, then the agent is done. If not, the agent must perform an action that brings him closer to the condition.

Notice that, 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 prevent his eventual achievement of 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.

3.2 Maintenance

An agent knows how to maintain a condition if he can continually and knowingly force it to be true, that is, 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.

A key difference with knowing how to achieve some condition is that achievement necessarily requires a bounded number of steps, whereas maintenance does not. 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. We shall finesse this aspect for simplicity.

If initially the agent does not know that p holds, then he clearly does not know how to maintain it. 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. The given agent must choose his action such that no combination of the other agents' actions can violate p. Not only must the agent's chosen action maintain p, it should also maintain his ability to maintain p further.

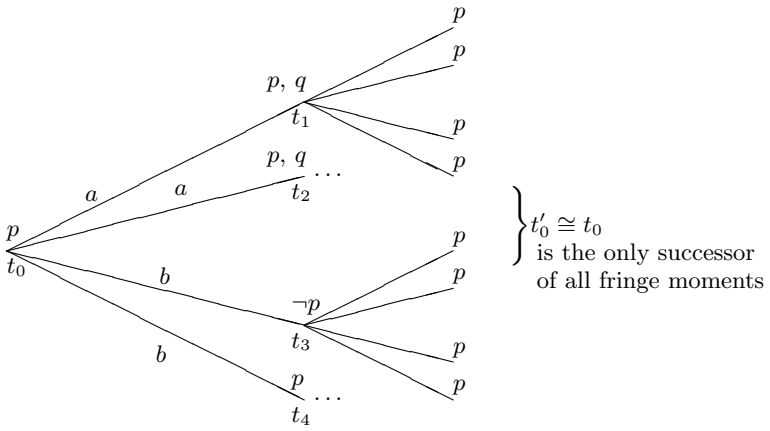


Figure 2: Maintenance

Example 13 Consider Figure 2. For simplicity assume that the agent has perfect knowledge, which means that the agent knows every proposition that is true in a given moment. We assume that t'_0, which has the same state and the congruent successor moments 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 after which he can still maintain p. From t_1, the choice of action is irrelevant, since the successors of t_1 are all succeeded by t_0. Similarly, t_2 is succeeded by t_0. Thus the agent can just cycle through t_0. The agent cannot maintain p in t_3, but can maintain p in all successors of t_3.

3.3 Putative Definitions of Maintenance

We consider some strawman definitions, and argue why they are intuitively unacceptable. This is anecdotal evidence that maintenance cannot be reduced to more common concepts and must be formalized independently. It is impossible to establish the above claim formally, because we are trying to evaluate formal definitions with respect to our intuitive understanding of maintenance.

Maintenance is not the formal dual of know-how to achieve p. Since maintenance resembles traditional safety properties, one might wonder if it is a dual of the corresponding achievement, that is, liveness, concept [11]. But it is not so. Just because an agent does not know how to achieve $\neg p$ does not entail that other agents do not know how to achieve $\neg p$. Thus if the other agents exercise their know-how, the given agent is unable to maintain p.

Maintenance is not the know-how to achieve $\mathsf{AG}p$. As discussed in Example 7, $\mathsf{AG}p$ means that p holds in all future moments (of the given moment). 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. Many interesting conditions fall into the latter category.

Maintenance is not the know-how to achieve $\mathsf{EG}p$. $\mathsf{EG}p$ means that there is a good path, but it does not entail that the agent will be able to prevent a bad path: the agent's actions may not be sufficiently 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 path where p holds forever.

4 Formalization

We formalize maintenance using trees of actions. We next define trees, use them to give a formal semantics for maintenance, and characterize it recursively.

4.1 Trees of actions

A tree of actions of an agent can be (a) empty, or (b) a single action (its *radix*) adjoined with a set of subtrees (each with 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 branches to have the same depth. We formalize the structure and depth of trees in mutual inductive definitions.

Definition 14 Υ, the set of trees, is defined as follows:

(a) $\bot \in \Upsilon$, where $\bot$ is the empty tree.

(b) $\langle a; \tau_1, \ldots, \tau_m \rangle \in \Upsilon$, where $a \in \mathcal{A}_c$, $\tau_1, \ldots, \tau_m \in \Upsilon$, and depth($\tau_1$)= ... = depth($\tau_m$). The trees $\tau_1, \ldots, \tau_m$ must have distinct radices, unless $m = 1$ and $\tau_1 = \perp$.

Definition 15 The depth of trees is defined as follows:

- depth($\perp$) $\stackrel{\text{def}}{=}$ 0.
- depth($\langle a; \tau_1, \ldots, \tau_m \rangle$) $\stackrel{\text{def}}{=}$ 1+depth(τ_1).

For simplicity, we write a instead of $\langle a; \perp \rangle$. Notice that trees can defined purely in terms of action symbols. However, a tree can be executed only if its actions are aligned with the available actions at the given moment.

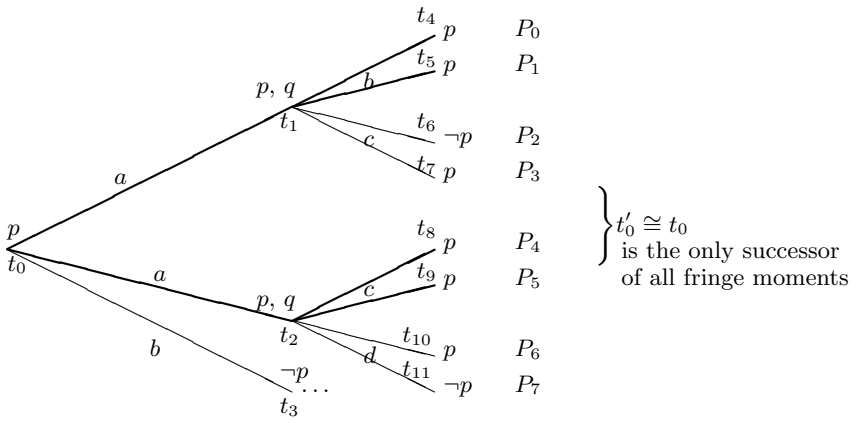


Figure 3: Maintenance over Trees of Actions

Figure 3 adds some annotations to Figure 2 in terms of the paths from t_0.

Example 16 Considering the maintenance of p in Figure 3, a tree at t_1 is $\langle b; \perp \rangle$ or b. A tree at t_2 is c. Similarly, composing the two we get $\langle a; b, c \rangle$ as a tree at t_0. This is highlighted.

4.2 Semantics of Maintenance

An agent maintains p over an empty tree if he knows that p holds currently. He maintains p over a single action, a, if he knows that he can perform a in the given state and p holds where a begins and where it ends. An agent maintains p over a general tree if he maintains it over its initial action and then over some applicable subtree.

Example 17 In Figure 3, assume that the agent has perfect knowledge. Then the agent can maintain p at moments t_8 and t_9 because he knows p there, that is, through the tree $\perp$. At t_2, the agent can maintain p using the tree c and at t_1 using the tree b. At t_0, the agent maintains p using the tree $\langle a; b, c \rangle$.

In order to give the formal semantics of maintenance, we define $[\![\tau]\!]_{t,p}$, the *maintenance denotation* of a tree τ, as the set of periods beginning at t over which p is maintained by τ. These are the periods over which the agent can knowingly select the right actions. $[\![\tau]\!]_{t,p} = \{\,\}$ means that p cannot be maintained using τ.

Definition 18 The *maintenance-intension* of a tree is defined as follows:

- $[\![\bot]\!]_{t,p} \stackrel{\text{def}}{=}$ (if $M \models_t \mathsf{K}p$, then $\{[P;t,t]\}$ else $\{\,\}$)
- $[\![a]\!]_{t,p} \stackrel{\text{def}}{=} \{[P;t,t'] : [P;t,t'] \in [\![a]\!]$ and $M \models_t \mathsf{K}p$ and $(\forall t_k : (t,t_k) \in \mathbf{K}(x) \Rightarrow (\exists P_k, t'_k : [P_k;t_k,t'_k] \in [\![a]\!]$ and $(\forall P_k, t'_k : [P_k;t_k,t'_k] \in [\![a]\!] \Rightarrow M \models_{t'_k} \mathsf{K}p)))\}$
- $[\![\langle a;\tau_1,\ldots,\tau_m\rangle]\!]_{t,p} \stackrel{\text{def}}{=} \{[P;t,t''] : (\exists t', j : [P;t,t'] \in [\![a]\!]_{t,p}$ and $[P;t',t''] \in [\![\tau_j]\!]_{t',p})$ and $(\forall t_k : (t,t_k) \in \mathbf{K}(x) \Rightarrow (\exists P_k, t'_k : [P_k;t_k,t'_k] \in [\![a]\!]_{t_k,p}$ and $(\forall P_k, t'_k : [P_k;t_k,t'_k] \in [\![a]\!]_{t_k,p} \Rightarrow (\exists t''_k, j : [P_k;t'_k,t''_k] \in [\![\tau_j]\!]_{t'_k,p})))))\}$

In other words, the agent maintains p over $[P;t,t'']$ iff the agent knows at t that he will maintain p over a, that is, till t', and then maintain p till t'' using some subtree. Notice that if a tree has a radix that is not an available action at a given moment, then its maintenance-intension at that moment would be empty.

Example 19 We can determine the maintenance-intension of the tree of Example 17. $[\![\langle a;b,c\rangle]\!]_{t_0,p} = \{[P_0;t_0,t_4],\ [P_1;t_0,t_5],\ [P_4;t_0,t_8],\ [P_5;t_0,t_9]\}$.

Lemma 20 shows that maintenance-intensions are stable in that if they are nonempty, they contain all of the alternatives at the moments epistemically related to the given moment.

Lemma 20 $([\![a]\!]_t \cap [\![a]\!]_{t,p} \neq \emptyset) \Rightarrow (\forall t_k : (t,t_k) \in \mathbf{K}(x) \Rightarrow \emptyset \subset [\![a]\!]_{t_k} \subseteq [\![a]\!]_{t_k,p})$ ∎

An agent maintains p to depth i if there is a tree of depth i over which he maintains p. An agent maintains p if he maintains it to all depths.

M12. $M \models_t \mathsf{K_m}^i p$ iff $(\exists \tau : \text{depth}(\tau) = i$ and $[\![\tau]\!]_{t,p} \neq \{\,\})$

M13. $M \models_t \mathsf{K_m}p$ iff $(\forall i : M \models_t \mathsf{K_m}^i p)$

Example 21 Following Example 17, we can verify that $\mathsf{K_m}^0 p$ holds at t_0, t_1, t_2, t_4, and t_5. Similarly, $\mathsf{K_m}^1 p$ holds at t_0, t_1, and t_2, because of trees a, b, and c, respectively. Also, $\mathsf{K_m}^2 p$ holds at t_0 because of the tree $\langle a;b,c\rangle$. Assuming that $t'_0 \cong t_0$ is the sole successor for moments t_4–t_{11}, we also have that $\mathsf{K_m}p$ holds at t_0, t_1, t_2, t_4, t_5, and t_7–t_{10}.

4.3 Recursive Characterization

Now we present a recursive characterization of maintenance. This characterization is used in Section 5, where we develop an approach based on the mu-calculus for computing know-how and maintenance.

Lemma 22 $\mathsf{K}p \wedge (\bigvee \alpha : \mathsf{K}(\mathsf{E}\langle\alpha\rangle\mathsf{true} \wedge \mathsf{A}[\alpha]\mathsf{K_m}^i p)) \equiv \mathsf{K_m}^{i+1} p$ ∎

Proof The base case (for $i = 0$) follows from the definition of $[\![a]\!]_{t,p}$. The inductive case follows from the definition of $[\![\langle a; \tau_1, \ldots, \tau_m\rangle]\!]_{t,p}$.

Theorem 23 $\mathsf{K}p \wedge (\bigvee \alpha : \mathsf{K}(\mathsf{E}\langle\alpha\rangle\mathsf{true} \wedge \mathsf{A}[\alpha]\mathsf{K}_\mathsf{m}p)) \equiv \mathsf{K}_\mathsf{m}p$ ∎

Proof This follows from the definition of $\mathsf{K}_\mathsf{m}p$ and Lemma 22.

The above characterization reflects the intuition of the formal definition of maintenance. Its conjuncts reflect the cases of the definition of maintenance intensions using which maintenance was formalized above.

4.4 Achievement

We now introduce the semantics for know-how with slight modifications from [26]. An agent achieves p over an empty tree if he knows that p holds currently. He achieves p over a single action, a, if he knows that he can perform a in the given state and p holds along any branch where a ends. An agent achieves p over a general tree if he knows that he can perform its initial action at the end of which (a) p will hold or (b) he will know which subtree to execute.

Example 24 In Figure 3, assume that the agent has perfect knowledge. Then the agent can achieve q at moments t_1 and t_2 because he knows q there, that is, through the tree $\bot$. At t_0, the agent can achieve q using the tree a.

In order to give the formal semantics of achievement, we define $(\!|\tau|\!)_{t,p}$ as the set of periods corresponding to the execution of τ, which begin at t and end at a moment at which p is achieved. $(\!|\tau|\!)_{t,p} = \{\,\}$ means that p cannot be achieved using τ.

Definition 25 The *achievement-intension* of a tree is defined as follows:

- $(\!|\bot|\!)_{t,p} \stackrel{\text{def}}{=}$ (if $M \models_t \mathsf{K}p$, then $\{[P; t, t]\}$ else $\{\,\}$)
- $(\!|\langle a; \tau_1, \ldots, \tau_m\rangle|\!)_{t,p} \stackrel{\text{def}}{=} \{[P; t, t''] : (\forall t_k : (t, t_k) \in \mathbf{K} \Rightarrow\ (\exists P_k, t'_k : [P_k; t_k, t'_k] \in [\![a]\!]_{t_k,p}$ and $(\forall P_k, t'_k : [P_k; t_k, t'_k] \in [\![a]\!]_{t_k,p} \Rightarrow\ (\exists t''_k, j : [P_k; t'_k, t''_k] \in (\!|\tau_j|\!)_{t'_k,p}))))\}$

In other words, the agent achieves p over $[P; t, t'']$ with respect to a tree of the form $\langle a; \tau_1, \ldots, \tau_m\rangle$ iff the agent knows at t that he will either (a) achieve p over a, or (b) after a, will achieve p (till t'') using some τ_i.

Example 26 We can determine the achievement-intension of the tree of Example 24. $(\!|a|\!)_{t_0,p} = \{[P_0; t_0, t_1],\ [P_1; t_0, t_2]\}$.

Recall that $x\mathsf{K}_\mathsf{a}p$ formalizes that the agent x knows how to achieve p. Our semantic definition of $\mathsf{K}_\mathsf{a}p$ is as follows. An agent achieves p if there is a tree over which he achieves p.

M14. $M \models_t \mathsf{K}_\mathsf{a}p$ iff $(\exists \tau : (\!|\tau|\!)_{t,p} \neq \{\,\})$

We now give a recursive characterization of achievement, which resembles the one given above for maintenance,and captures essentially the same intuition.

Theorem 27 $\mathsf{K}p \vee (\bigvee \alpha : \mathsf{K}(\mathsf{E}\langle\alpha\rangle\mathsf{true} \wedge \mathsf{A}[\alpha]\mathsf{K}_\mathsf{a}p)) \equiv \mathsf{K}_\mathsf{a}p$ ∎

5 Computing Decision Graphs

One may use a formalization of a concept in various ways, such as in automatic or guided theorem provers, and tools for checking prespecified models. The latter approach, which we adapt,has proved particularly effective in program verification [8]. We discuss maintenance at length and then show how achievement can be similarly handled. Given a model, describing moments and actions among them, and a formula, describing a goal, we compute the set of moments at which the formula holds. The model may be implicitly specified without enumerating all the moments. In fact, our approach incrementally builds a models that verifies the given formula. The result encodes the actions that must be performed in each moment to realize the given goal.

In our framework, the (maintenance) planning problem is posed in terms of the following three components:

- t_0, an initial state
- p, a condition to be maintained (or, more generally, any goal expressed in our formal language)
- M, a means of generating a model.

Recall that the denotations or intensions of formulae in $\mathcal{ACTL}^*$ are sets of moments. The fixpoint approach is based on a lattice of denotations. This lattice uses as its partial order the standard set-inclusion ordering on sets of moments. Intuitively, we begin with the most permissive assignment, namely, one reflecting the assumption that a formula can be maintained everywhere. We then prune the set of moments based on whether the base case of knowledge about the given proposition obtains. From this reduced set of moments, we recursively prune moments at which actions cannot be selected that would guarantee maintenance, until a fixpoint is attained. The prune operation is based on the monotonic functional we define below. The fixpoint attained is the greatest fixpoint of this functional and yields the desired answer.

The above approach naively generates all states, although several of them would not be needed for the final plan. We later show how we can avoid generating useless states, thus leading to greater efficiency. We now turn to the formal development.

5.1 The Propositional Mu-Calculus

Our presentation below is self-contained. It has been simplified to include just the features we need. Let $\mathcal{ACTL}^{*\mu}$ be $\mathcal{ACTL}^*$ extended to accommodate the mu-calculus. Let Ξ be a set of propositional variables. The operators μ (mu) and ν (nu) bind propositional variables. Notice that we could replace the temporal operators by their mu-calculus abbreviations, but for simplicity we leave them in.

L6. Replace $\mathcal{ACTL}^*$, $\mathcal{ACTL}^*{}_\pi$ by $\mathcal{ACTL}^{*\mu}$ $\mathcal{ACTL}^{*\mu}_\pi$ in the syntax rules of Section 2

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

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

Definition 28 A *functional* on $Z \in \Xi$ is an expression free in Z in which Z occurs inside an even number of negations. Functionals are typically notated with $\zeta(Z)$ with subscripts.

The above entails that $\zeta(Z)$ is monotone in Z and, by the Tarski-Knaster theorem, has both a greatest and a least fixpoint.

For $p \in \mathcal{ACTL}^*$, the semantics of Section 2.3 holds—the definitions below lift $[\![\,]\!]$ from atomic propositions to all of $\mathcal{ACTL}^*$. $[\![\,]\!]_\pi$ is the path version of $[\![\,]\!]$.

Semantically, the propositional variables are treated just like propositions except that their interpretation is not fixed in the model, but rather given explicitly. Thus $M \models_t Z$ iff $t \in [\![Z]\!]$, where $[\![Z]\!]$ is the denotation of Z. The functionals in $\mathcal{ACTL}^{*\mu}$ are functions from sets of moments to sets of moments.

Definition 29 Let $f_\zeta : \wp(\mathbf{T}) \mapsto \wp(\mathbf{T})$ be a function derived from the syntax and semantics of $\mathcal{ACTL}^{*\mu}$. Given a set of moments $S \subseteq \mathbf{T}$, $f_\zeta(S) = [\![\zeta(Z)]\!]$, where $[\![Z]\!] = S$.

Intuitively, f_ζ is the function corresponding to the expression ζ. $(\mu Z : \zeta(Z))$ and $(\nu Z : \zeta(Z))$ are, respectively, the least and greatest fixpoints of the functional $\zeta(Z)$. By the Tarski-Knaster theorem, $(\nu Z : \zeta(Z)) = \bigcap_{i \geq 0} \zeta^i(\mathsf{true})$ and $(\mu Z : \zeta(Z)) = \bigcup_{i \geq 0} \zeta^i(\mathsf{false})$. Here ζ^i denotes ζ composed i times.

M15. $[\![p]\!] = \{t : M \models_t p\}$, where $p \in \mathcal{ACTL}^*$

M16. $[\![p]\!]_\pi = \{P : P \in \mathbf{P}_t \text{ and } M \models_{P,t} p\}$, where $p \in \mathcal{ACTL}^*{}_\pi$

M17. $[\![\zeta]\!] = f_\zeta$

M18. $[\![(\mu Z : \zeta(Z))]\!] = \bigcup_{i \geq 0} [\![\zeta^i(\mathsf{false})]\!]$

M19. $[\![(\nu Z : \zeta(Z))]\!] = \bigcap_{i \geq 0} [\![\zeta^i(\mathsf{true})]\!]$

Example 30 We have the equation $\mathsf{AG}p \equiv p \wedge (\bigwedge \alpha : \mathsf{A}[\alpha]\mathsf{AG}p)$. This merely states that p holds at all moments in the future of a given moment, t, iff p holds at t, and p holds in the future of all successor moments of t (after any action). This idea can be captured in the mu-calculus with the equation $\mathsf{AG}p \equiv (\nu Z : p \wedge (\bigwedge \alpha : \mathsf{A}[\alpha]Z))$, where Z's denotation ranges over $\wp(\mathbf{T})$.

For simplicity, we assume that boolean, temporal, and knowledge expressions are calculated by a subroutine.

5.2 Maintenance in the Mu-Calculus

Given a formula p, we define a functional ζ_m (which incorporated p) to capture maintenance. Here C refers to control necessitation as given in Definition 12.

Definition 31 $\zeta_\mathsf{m}(Z) \stackrel{\text{def}}{=} \mathsf{K}p \wedge \mathsf{C}Z$

This functional is obvious from the characterization given in Section 4.3. Lemma 32 states that $\zeta_\mathsf{m}(Z)$ is monotonic in Z.

Lemma 32 $Z_1 \subseteq Z_2 \Rightarrow \zeta_{\mathsf{m}}(Z_1) \subseteq \zeta_{\mathsf{m}}(Z_2)$ ∎

Thus $\zeta_{\mathsf{m}}(Z)$ must have a greatest and a least fixpoint. The least fixpoint is $\{\}$ and is uninteresting. However, Theorem 33 states that the greatest fixpoint of $\zeta_{\mathsf{m}}(Z)$ equals $\mathsf{K}_{\mathsf{m}}p$.

Theorem 33 $\mathsf{K}_{\mathsf{m}}p \equiv (\nu Z : \zeta_{\mathsf{m}}(Z))$ ∎

Proof Theorem 23 indicates that $\mathsf{K}_{\mathsf{m}}p$ is a fixpoint of $\zeta_{\mathsf{m}}(Z)$. Let Z be some fixpoint of $\zeta_{\mathsf{m}}(Z)$. Because $\mathsf{K}p$ holds everywhere in Z, it is clear that ${\mathsf{K}_{\mathsf{m}}}^0 p$ holds everywhere in Z. Therefore, using the actions (selected by the existential quantifier in $\mathsf{C}Z$) that occur to make $\mathsf{C}Z$ true, we can inductively construct trees of depths 1, 2, ..., using ${\mathsf{K}_{\mathsf{m}}}^0 p$ as the base case. Thus, for all depths i, ${\mathsf{K}_{\mathsf{m}}}^i p$ holds everywhere in Z. Hence, $\mathsf{K}_{\mathsf{m}}p$ holds everywhere in Z. Consequently, $Z \subseteq [\![\mathsf{K}_{\mathsf{m}}p]\!]$. In other words, $\mathsf{K}_{\mathsf{m}}p$ includes every fixpoint of $\zeta_{\mathsf{m}}(Z)$. Therefore, $\mathsf{K}_{\mathsf{m}}p$ is the greatest fixpoint of $\zeta_{\mathsf{m}}(Z)$.

5.2.1 Naive Approach

A naive algorithm for computing the above formula follows directly from the definition of greatest fixpoints. 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 $j \leq |\mathbf{T}|$. Thus termination is guaranteed if $\mathbf{T}$ is finite.

Example 34 Consider Figure 3 again. Let $\mathbf{T} = \{t_0 \ldots t_{11}\}$—that is, identify t_0 with t_0'. We wish to compute $[\![\mathsf{K}_{\mathsf{m}}p]\!]$. Initially, $[\![\mathsf{true}]\!] = \{t_0 \ldots t_{11}\}$. And, $[\![\zeta_{\mathsf{m}}(\mathsf{true})]\!] = \{t_0 \ldots t_2, t_4 \ldots t_{10}\}$ (t_3 and t_{11} are eliminated). No further moments are eliminated, that is, $[\![\zeta_{\mathsf{m}}^2(\mathsf{true})]\!] = [\![\zeta_{\mathsf{m}}(\mathsf{true})]\!]$. Hence, we have found the fixpoint, which is the desired answer for $[\![\mathsf{K}_{\mathsf{m}}p]\!]$. From this we can derive the maintenance plan as an infinite decision tree rooted at t_0, finitely expressed as a decision graph $\tau = \langle a; \langle b, a, \#\tau\rangle, \langle c, a, \#\tau\rangle\rangle$, where $\#\tau$ refers back to τ.

5.2.2 Control Progression

The above, naive approach computes $[\![(\nu Z : \zeta_{\mathsf{m}}(Z))]\!]$, and in the process also computes all of $[\![\mathsf{true}]\!]$, only to check if t_0 is part of the answer. This is obviously too wasteful. A better approach can be designed that directly tests whether $t_0 \in [\![(\nu Z : \zeta_{\mathsf{m}}(Z))]\!]$, and computes only the states necessary to make that determination. This is based on the following definitions and results, which intuitively capture the idea that we only need to explore the moments that result from an action in t_0. Further, we need explore the successors of only those moments that remain candidates for maintenance—moments where maintenance fails can be pruned immediately.

We define $\beta(z, Z)$ as the set of all minimal subsets Y of Z, such that $z \in [\![\zeta_{\mathsf{m}}(Y)]\!]$ guarantees that $z \in [\![\zeta_{\mathsf{m}}(Z)]\!]$. Thus $\beta(z, Z)$ is the set of minimal certificates for $z \in [\![\zeta_{\mathsf{m}}(Z)]\!]$. The point of defining $\beta(z, Z)$ is that each of its member sets contains moments that are relevant for evaluating $\mathsf{K}_{\mathsf{m}}p$, and no more—typically, the set of relevant moments would be much smaller than $[\![\mathsf{true}]\!]$.

But how can we construct $\beta(z, Z)$? Given our intuitions about the role of control necessitation in the definition of maintenance, each member of $\beta(z, Z)$ should be a

set of moments that result from an action performed in z. For this reason, we define $\gamma(a, z)$ as the post-image of action a. Next, we use γ to define β.

Definition 35 $\gamma(a, z) \stackrel{\text{def}}{=} \{t : (\exists P : [P; z, t] \in [\![a]\!])\}$

Definition 36 $\beta(z, Z) \stackrel{\text{def}}{=} \{Y : Y \subseteq Z \text{ and } (\exists a : Y = \gamma(a, z)) \text{ and } (\forall b : \gamma(b, z) \not\subset Y)\}$

Thus, $\beta(z, Z)$ preserves the property that all or no outcomes of an action are included in any of its member sets of moments—this prevents declaring success erroneously. Lemma 37 follows from the monotonicity of ζ_{m}.

Lemma 37 $\beta(z, Z) \neq \{\,\}$ iff $z \in [\![\zeta_{\mathsf{m}}(Z)]\!]$ ∎

Similarly, $z \in [\![\zeta_{\mathsf{m}}^{i+1}(Z)]\!]$ iff $\beta(z, \zeta_{\mathsf{m}}^{i}(Z)) \neq \{\,\}$. Computing $\beta(z, \zeta_{\mathsf{m}}^{2}(Z))$ requires computing the post-images of the available actions when performed in the states of the post-images of the actions performed in z. However, these post-images are determined as part of $\beta(z, Z)$. This process is inductive. We term it *control progression*, because it behaves like temporal progression in determining successor states, but takes into account the control necessitation inherent in correct decision graphs.

5.2.3 Algorithm to Compute Decision Graphs

The above observations yield the following procedure for determining a decision graph from a state z and an active set of moments Z. Importantly, Z is specified implicitly—only those members of it that immediately affect whether $z \in [\![\zeta_{\mathsf{m}}(Z)]\!]$ are made explicit.

First compute $\beta(z, Z)$. If $\beta(z, Z)$ is empty, return failure. If $\beta(z, Z)$ is nonempty, construct an AND-OR graph with z as an OR node, and each member of $\beta(z, Z)$ as an AND node. Since each member of $\beta(z, Z)$ corresponds to the post-image of some action, the edge from z to an AND node is labeled by the action whose post-image is captured by the AND node.

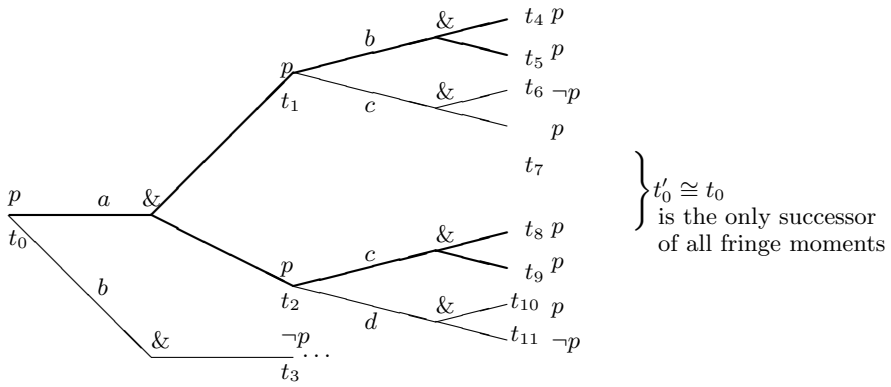
Inductively, determine whether $y \in [\![\zeta_{\mathsf{m}}(Z)]\!]$ for each state y in an AND node. If it fails, delete the entire AND node. When all AND successors of an OR node are deleted, delete it as well. Since we identify moments that represent the same state, the computation need not be repeated for states that are previously explored. The procedure essentially involves constructing and searching the AND-OR graph. As usual, it helps to search depth first to reduce the space requirements. The program given in Figure 4 is invoked as `maint(`t_0`, { }, { }, `p`)`.

Example 38 We redo Example 34 by invoking our algorithm from t_0. Figure 5 illustrates this case. As before, for simplicity we assume perfect knowledge. In the first iteration, the algorithm selects action a and the moments t_1 and t_2; the moment t_3 and its successors are pruned. From t_1, the algorithm identifies action b and moments t_4 and t_5. From t_2, it identifies action c and moments t_8 and t_9. Thus, it effectively computes the result of Example 19. From each of the moments t_4, t_5, t_8, and t_9, action a may be chosen to get to t_0, or rather to $t_0' \cong t_0$.

```
maint(node, graph, explored, p)
     if node ∈ graph return (node, graph, explored)
     if node ∈ explored return (nil, nil, explored)
     if node ⊭ Kp return (nil, nil, explored ∪ {node})
     beta ← {(a, Y): Y is the post-image of a in node}
     g ← graph
     e ← explored
     forall (a, Y) ∈ beta {
        (nY, gY, eY) ← and_maint(Y, g, e, p)
        g ← splice(g, node, a, nY, gY) //equals g if gY = nil
        e ← eY
        }
     if no out-edge from node return (nil, nil, e)
     return (node, g, e)

and_maint(Y, graph, explored, p)
   and_node ← new And_Node
   g ← graph
   e ← explored
   forall n ∈ Y {
      (nn, gn, en) ← maint(n, g, e, p)
      if nn = nil return (nil, nil, en)
      insert_into(and_node, nn)
      g ← gn
      e ← en
      }
   return (and_node, g, e)
```

Figure 4: Maintenance by Control Progression



(& indicates an AND node)

Figure 5: Control Progression Illustrated (Final graph in bold)

5.3 Achievement

Given the above development, the case of achievement is quite straightforward. We define a functional ζ_{a} to capture the achievement of p:

Definition 39 $\zeta_{\mathsf{a}}(Z) \stackrel{\text{def}}{=} \mathsf{K}p \vee \mathsf{C}Z$

This functional is obvious from the characterization of Section 4.4. Theorem 40 states that the least fixpoint of $\zeta_{\mathsf{a}}(Z)$ equals $\mathsf{K}_{\mathsf{a}}p$.

Theorem 40 $\mathsf{K}_{\mathsf{a}}p \equiv (\mu Z : \zeta_{\mathsf{a}}(Z))$ ∎

The computation of the corresponding decision graph proceeds along similar lines to the above.

6 Discussion

We showed how a logics of program approach can be adapted for AI planning, yielding more power than previous approaches. The similarities between AI planning and logics of program are known, but have not been fully exploited. Ultimately, we believe, this synergy must be exploited for building robust intelligent systems. Our approach

- uses logics of program tools for AI ends
- is formal and rigorous
- allows a richer variety of goals than traditional approaches
- involves algorithms that exploit the special structure of the computations, as reflected in the syntax of the mu-calculus formulation.

The mu-calculus enables declarative descriptions of complex computations through fixpoints. Fixpoints are well-suited to describing important concepts from planning. An expanding body of research exists to construct efficient tools for the mu-calculus and TL, which can be marshaled for planning. Recent examples include Bhat & Cleaveland [4], Clarke *et al.* [8], and Holzmann [14].

Our approach is orthogonal to how one determines the formulae true in each state in the post-image of an action, but one can assume a theory of actions with fully specified effects for this purpose, such as that of Schubert [24], with some enhancements to allow nondeterminism. The problem of modeling concurrent events and their effects was also considered in an abstract setting by Georgeff, who developed an approach based on persistence and causality [13]. More recent approaches, for example, by Baral [2] and Thielscher [28], seek to characterize the effects of actions in different circumstances.

Complexity. The complexity of the above approach has not been carefully studied. It relies on having a finite set of moments to explore. Termination requires identifying moments with those of the same state (that is, the same atomic propositions and knowledge of the agent) that have been visited. The number of iterations if bounded by the length of the longest path in $\mathbf{T}$, where length is given by the number of actions.

Literature. There has been an enormous amount of research on various kinds of planning. We mention only some selected works relating TL and planning. (TL approaches to be distinguished from temporal reasoning in general, which all planners must perform to some degree.)

One of the first applications of classical techniques in planning was identified by Georgeff [12]. Georgeff proposed *process models* to represent how the actions were selected by different agents. He also suggested the use of model-based as opposed to axiomatic techniques for reasoning about the relationships among process models.

Singh [26] and Belnap & Perloff [3] define operators, but do not develop algorithms to compute them. The present approach shows how algorithms can be derived from the semantics of those operators.

Bacchus & Kabanza define goals as sequences of states, and develop an algorithm to produce plans that satisfy such goals [1]. The heart of their approach is a progression algorithm, which given formulae for a state, derives formulae for the next state. This assumes that actions are deterministic, and changes to the environment are brought about solely by the agent (p. 1217). In a fundamental sense, we capture intuitions similar to temporal progression, but extend them to branching, nondeterministic models. Bacchus & Kabanza allow metric goals, which we believe can be added to our approach as well.

Pain-Barre [20] combines heuristic planning with a deductive approach to world changes due to actions. This approach also involves depth-first search, but focuses exclusively on achievement goals. Dengler also uses TL to go beyond traditional achievement planners, but he focuses on enhancing the interactivity of the planner with a human user [10]. This approach is quite sophisticated in considering the stepwise refinement of plans, although the underlying framework is linear. We believe it could be fruitfully combined with our approach.

Another use of TL in planning involves modeling actions as procedural programs in the style of Algol, for instance, and using conventional TL techniques for proving properties about them. Biundo & Stephan [5] and Łukaszewicz & Madalińska-Bugaj [17] develop such approaches.

Singh introduced the mu-calculus for reasoning about action and developed a model-checking approach [25]. Independently, De Giacomo & Chen [9] give a mu-calculus characterization of plans, which however is based on a process algebra reminiscent of Milner's CCS [19], rather than an explicit branching-time logic as in the present approach. Both of the above approaches assume a fully specified model, instead of constructing one incrementally as here.

Future Directions. The technique proposed here generalizes mu-calculus techniques for temporal and dynamic logic operators to handle interesting forms of AI planning. Although this technique is only the beginning, it hints at more what can be accomplished. We are studying syntax-directed techniques for generating models and plans from the common kinds of mu-calculus expressions that arise in planning problems. Another challenge is to consider more complex goal specifications than allowed above.

We have taken some early steps in formalizing plans for multiagent environments. Much of our technical development considers the plans of a single agent, but by allowing branching and nondeterminism, it enables planning in multiagent environments. In this way, it is similar in style to [12, 13]. We leave it to future research to formalize ideas from cooperative planning to the same level of detail as carried out above.

Acknowledgments

The author benefited greatly from some early discussions with Allen Emerson, and from meticulous comments by the anonymous reviewers.

References

[1] Fahiem Bacchus and Froduald Kabanza. Planning for temporally extended goals. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1215–1222, 1996.

[2] Chitta Baral. Reasoning about actions: Non-deterministic effects, constraints, and qualification. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 2017–2023, 1995.

[3] Nuel Belnap and Michael Perloff. Seeing to it that: A canonical form for agentives. *Theoria*, 54(3):175–199, 1988.

[4] Girish Bhat and Rance Cleaveland. Efficient model checking via the equational mu-calculus. In *Proceedings of the 11th International Symposium on Logic in Computer Science*, pages 304–312, 1996.

[5] Susanne Biundo and Werner Stephan. Modeling planning domains systematically. In *Proceedings of the European Conference on Artificial Intelligence*, pages 599–603, 1996.

[6] Mark A. Brown. Action and ability. *Journal of Philosophical Logic*, 19:95–114, 1990.

[7] Brian F. Chellas. Time and modality in the logic of agency. *Studia Logica*, 51(3/4):485–517, 1992.

[8] E. Clarke, O. Grumberg, and D. Long. Model checking. In *Proceedings of the International Summer School on Deductive Program Design*, pages 428–439, 1990.

[9] Giuseppe De Giacomo and Xiao Jun Chen. Reasoning about nondeterministic and concurrent actions: A process algebra approach. In *Proceedings of the National Conference on Artificial Intelligence*, pages 658–663, 1996.

[10] Dietmar Dengler. Customized plans transmitted by flexible refinement. In *Proceedings of the European Conference on Artificial Intelligence*, pages 609–613, 1996.

[11] E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. North-Holland, Amsterdam, 1990.

[12] Michael P. Georgeff. A theory of action for multiagent planning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 121–125, 1984.

[13] Michael P. Georgeff. The representation of events in multi-agent domains. In *Proceedings of the National Conference on Artificial Intelligence*, pages 70–75, 1986.

[14] Gerard J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–296, May 1997.

[15] David Kinny and Michael P. Georgeff. Modelling and design of multi-agent systems. In *Intelligent Agents III: Agent Theories, Architectures, and Languages*, pages 1–20, 1997.

[16] Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[17] Witold Łukaszewicz and Ewa Madalińska-Bugaj. Reasoning about action and change using Dijkstra's semantics for programming languages: Preliminary report. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1950–1955, 1995.

[18] Rob Miller. A case study in reasoning about actions and continuous change. In *Proceedings of the European Conference on Artificial Intelligence*, pages 624–628, 1996.

[19] Robin Milner. *Communication and Concurrency*. Prentice-Hall International, Hemel Hempstead, UK, 1989.

[20] Cyril Pain-Barre. DEDAL: A deductive and algorithmic planning system. In *Proceedings of the European Conference on Artificial Intelligence*, pages 629–633, 1996.

[21] Anand S. Rao and Michael P. Georgeff. Asymmetry thesis and side-effect problems in linear-time and branching-time intention logics. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 498–504, 1991.

[22] Anand S. Rao and Michael J. Wooldridge, editors. *Foundations and Theories of Rational Agency*. Applied Logic Series. Kluwer, Dordrecht, 1998.

[23] Raymond Reiter. Natural actions, concurrency and continuous time in the situation calculus. In *Proceedings of the Third Symposium on Logical Formalizations of Commonsense Reasoning*, 1996.

[24] Lenhart Schubert. Monotonic solution of the frame problem in the situation calculus: An efficient method for worlds with fully specified actions. In Henry E. Kyburg, Ron Loui, and Greg Carlson, editors, *Knowledge Representation and Defeasible Reasoning*, pages 23–67. Kluwer, 1990.

[25] Munindar P. Singh. Maintenance and prevention: Formalization and fixpoint characterization. In *Proceedings of the ECAI Workshop on Logic and Change*, August 1994.

[26] Munindar P. Singh. *Multiagent Systems: A Theoretical Framework for Intentions, Know-How, and Communications*. Springer-Verlag, Heidelberg, 1994.

[27] Munindar P. Singh. Know-how. In *[22]*. 1998.

[28] Michael Thielscher. The logic of dynamic systems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1956–1962, 1995.

[29] Wiebe van der Hoek, Bernd van Linder, and John-Jules Ch. Meyer. A logic of capabilities. TR IR-330, Vrije Universiteit, Amsterdam, 1993.

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