FORMULATING MULTIAGENT, DYNAMIC-WORLD PROBLEMS IN THE CLASSICAL PLANNING FRAMEWORK

Edwin P.D. Pednault*
Aritificial Intelligence Center
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

ABSTRACT

This paper demonstrates how some multiagent, dynamic-world planning problems may be formulated as single-agent, static-world problems, thereby permitting them to be solved with techniques originally intended for the latter. This approach to formulating such problems requires that dynamic worlds be modeled in the same manner as their static couterparts, and that simultaneous actions be modeled as sequential actions. The former is accomplished by introducing time as a parameter in the description of dynamic worlds. To model simultaneous actions as sequential ones, the idea of a *boundary condition* is introduced. A boundary condition exists at a point in time and determines the future course of events, up to the next boundary condition. The effect of simultaneous actions is then modeled by modifying boundary conditions sequentially. To provide concrete examples, a new language is introduced for the purpose of describing actions and their effects. This language, called ADL, combines the the notational convenience of the STRIPS operator language with the semantics and expressive power of the situation calculus. ADL thus has the same facilities as the STRIPS language for coping with the frame problem, but it is a more expressive language in that it permits actions to be described whose effects change according to the circumstances underwhich they are performed. In addition, ADL overcomes the semantic pitfalls of the STRIPS language that are discussed by Lifschitz.

---

*Current address: AT&T Bell Laboratories, Crawfords Corner Road, Holmdel, New Jersey 07733.

1. INTRODUCTION

Until recently, virtually all work in automatic planning has been concerned with ways of representing and solving what might be called the *classical planning problems*. In this type of problem, the world is regarded as being in one of a potentially infinite number of states. Performing an action causes the world to pass from one state to the next. In the specification of a problem, we are given a set of goals, a set of allowable actions, and a description of the world's initial state. We are then asked to find a sequence of actions that will transform the world from any state satisfying the initial-state description to one that satisfies the goal description. This framework has typically been used to model real-world problems that involve a single agent (e.g., a robot) operating in an environment that changes only as the result of the agent's action, and otherwise remains static.

Much contemporary research, on the other hand, is concerned with broader classes of planning problems, particularly those that involve multiple agents operating in dynamic worlds; for example, two children playing on a seesaw. To model such problems, appropriate representations for simultaneous actions and continuous processes are required, along with mechanisms for solving problems by using such representations. Hendrix [15] was one of the first to address these representational issues, having developed a system for simulating the effects of simultaneous actions in a dynamic world. More recent contributions have been made by Allen [2], Georgeff [9–12], Lansky [16, 17], and McDermott [21, 23], each of whom has used a different approach to the problem of representing actions and processes: Allen advocates a logic of time intervals, Georgeff employs a modified state-based approach, Lansky uses an event-based logic, while McDermott prefers a logic that combines states with continuous time lines. Stuart [29, 30] has examined the problem of synchronizing plans executed by different agents to avoid undesirable interactions among plans. Cheeseman [4], Dean [6], Tate [5, 33], Vere [35, 36], and Vilain [37] have concentrated primarily on representing and reasoning about the time and duration of events.

At this stage in the game, the primary objective is to develop epistemologically adequate [20] formalisms for representing and reasoning about simultaneous actions and continuous processes. The next step is to construct planning techniques that utilize these formalisms. Some work has already been done in this direction, notably by Allen and Kooman [1], Cheeseman [4], Georgeff and Lansky [10], Lansky [16], and McDermott [22]. However, these techniques are still in their infancy. The purpose of this paper is to help

further these efforts by showing how some multiagent, dynamic-world problems may be formulated and solved in the classical planning framework. It is hoped that, by looking at the classical planning framework in this new light, it may be possible to discern how solution techniques developed for the classical problems can be transferred to frameworks that are more suitable for solving multiagent, dynamic-world problems.

This paper focuses primarily on ways of formulating multiagent, dynamic-world problems as classical planning problems. As part of our exposition, a new language, called ADL, will be introduced for the purpose of describing actions and their effects. ADL differs from its predecessors in that it combines the notational convenience of the STRIPS operator language [7] with the semantics and expressive power of the situation calculus [20] and first-order dynamic logic [13]. From the point of view of syntax, descriptions in ADL resemble STRIPS operators with conditional add and delete lists. This syntax permits actions to be described whose effects are highly state-dependent. At the same time, it allows the frame problem [14] to be circumvented to the extent that one need only specify what changes are made when an action is performed, not what remains unaltered. However, the semantics of ADL is drastically different from that of STRIPS. Whereas STRIPS operators define transformations on state descriptions, ADL schemas define transformations on the states themselves. In this respect, the semantics of ADL is similar to that of the situation calculus and first-order dynamic logic. Furthermore, by adopting a different semantics, ADL avoids the pitfalls of the STRIPS language that are discussed by Lifschitz [18].

As we shall see, ADL is well suited to the multiagent, dynamic-world problems that are presented. Once a problem is formulated with ADL, it can be solved by means of the techniques described in my thesis [25] and in a earlier technical report [24]. An explicative review of these techniques, however, is beyond the scope of this paper.

2. DEFINING THE CLASSICAL PLANNING PROBLEMS

To make clear the issues involved in formulating multiagent, dynamic-world problems as classical planning problems, the latter will now be defined mathematically. This definition is set-theoretic in nature and is based on ideas borrowed from first-order dynamic logic [13].

As previously stated, the world is regarded, in classical planning problems, as being in any one of a potentially infinite number of states. The effect of an action is to cause

the world to jump from one state to another. This permits an action to be characterized as a set of current-state/next-state pairs, summarizing the possible state transitions that could occur when the action is performed. Stated mathematically, an action is a set a of ordered pairs of the form $\langle s, t \rangle$, where s and t are states, s being the "current state" and t being the "next state."

In many situations, there is no uncertainty as to how the world will change when an action is performed. In such cases, performing an action will cause the world to jump from one state to a unique next state. Mathematically speaking, an action a has this property if and only if for every state s there is at most one state t such that $\langle s, t \rangle \in a$ (i.e., such that the ordered pair $\langle s, t \rangle$ appears in a). Hence, performing action a when the world is in state s will always cause the world to jump to state t.

In other situations, the effect of an action might be unpredictable. In such cases, there will be a collection of states that the world could jump to when the action is performed, but the exact state would not be known beforehand. In mathematical terms, the effect of performing action a in state s will be uncertain if and only if there is more than one state t such that $\langle s, t \rangle \in a$. For example, if performing action a in state s will cause the world to jump to one of the states $t_1, \ldots, t_n$, but it is not known beforehand which it will be, then this is modeled by having each of the ordered pairs $\langle s, t_1 \rangle, \ldots, \langle s, t_n \rangle$ appear in the set a.

Although some actions can be performed under any circumstances, most actions can occur only when the world is in one of a limited number of states. For example, to walk through a doorway, the door must be open. If the world is in a state in which the door is closed, the action of walking through the doorway becomes impossible. In our mathematical framework, an action a may be executed in a state s if and only if there is some state t such that $\langle s, t \rangle \in a$. The set of states in which a may take place is given by

$$\mathrm{dom}(a) = \{s \mid \langle s, t \rangle \in a \text{ for some state } t\}$$

The function dom is borrowed from set theory [27, 31], where $\mathrm{dom}(a)$ is called the *domain* of the set of ordered pairs a.

In the statement of a classical planning problem, we are given a set of allowable actions, a set of possible initial states, and a set of acceptable goal states. Told that the world is currently in one of the possible initial states, we are then asked to find a sequence

of allowable actions that, when executed, will leave the world in one of the acceptable goal states. A classical planning problem can therefore be viewed as an encoding of this information:

Definition 2.1. A *classical planning problem* is a quadruple of the form $\langle \mathcal{W}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, where

(1) $\mathcal{W}$ is the set of all possible states of the world
(2) $\mathcal{A}$ is the set of allowable actions
(3) $\mathcal{I}$ is the set of possible initial states
(4) $\mathcal{G}$ is the set of acceptable goal states.

A solution to a classical planning problem is a sequence of actions $a_1 a_2 \cdots a_n$ that will transform the world from any of the possible initial states into one of the acceptable goal states. To guarantee this result, a number of conditions must be satisfied. First, it must be possible to execute each of the actions in the given order. This requires that it be possible to first perform a_1 in any of the possible initial states, then a_2 in any state that could result from performing a_1, then a_3 in any state that could result from performing a_2, etc. If these conditions hold, the sequence is said to be *executable*. Finally, for the sequence to be a solution, the world must necessarily be in one of the goal states once all the actions have been carried out.

To express these conditions mathematically, the following notation is introduced.

(1) Let ϵ denote the empty sequence; that is, the sequence containing no actions. "Executing" ϵ therefore leaves the world undisturbed.

(2) If σ and γ are two sequences of actions, then $\sigma\gamma$ is the sequence obtained by appending γ to the end of σ. For example, if $\sigma = a_1 a_2$ and $\gamma = a_3 a_4 a_5$, then $\sigma\gamma = a_1 a_2 a_3 a_4 a_5$. Note that $\sigma\epsilon = \epsilon\sigma = \sigma$.

(3) If there exists a sequence γ such that $\sigma\gamma = \theta$, then σ is said to be a *prefix* of θ. For example, the prefixes of the sequence $a_1 a_2 a_3 a_4 a_5$ are ϵ, a_1, $a_1 a_2$, $a_1 a_2 a_3$, $a_1 a_2 a_3 a_4$, and $a_1 a_2 a_3 a_4 a_5$.

The concept of a prefix permits us to express the executability conditions for a sequence as follows: a sequence θ is executable if and only if, for every prefix σa of θ, where a is an

action, a can be performed in every state that could result from the execution of σ. The set of states that could result when σ is executed is given by the function Result(σ, I), which is defined recursively as follows:

$$\text{Result}(\epsilon, I) = I \tag{2.2a}$$

$$\text{Result}(\gamma a, I) = \{t \mid \langle s, t\rangle \in a \text{ for some } s \in \text{Result}(\gamma, I)\} \tag{2.2b}$$

Thus, if no actions task place, the set of possible resulting states is merely the set of possible initial states. If one or more actions are performed, the set of resulting states is obtained by considering every state transition that could possibly occur when the sequence is executed. Thus, an action a can be performed in every state that could result from the sequence σ if and only if Result$(\sigma, I) \subset \text{dom}(a)$ (i.e., every state in Result(σ, I) also appears in dom(a)). This permits the following definitions to be formulated:

Definition 2.3. A sequence of actions θ is said to be *executable* with respect to a set of possible initial states I if and only if Result$(\sigma, I) \subset \text{dom}(a)$ for every prefix σa of θ such that a is an action.

Definition 2.4. A *solution* to a classical planning problem $\langle \mathcal{W}, \mathcal{A}, I, \mathcal{G}\rangle$ is a sequence of actions θ drawn from $\mathcal{A}$ such that θ is executable with respect to I and Result$(\theta, I) \subset \mathcal{G}$.

3. THE ONTOLOGY OF ACTIONS IN CLASSICAL PLANNING

When a formalism is proposed for describing some aspect of the real world, two questions naturally arise: in what sense does the formalism reflect reality, and to what degree? This section is intended to at least partially answer these questions with respect to the classical planning problems. Specifically, we shall consider how various kinds of real-world planning problems may be modeled as classical planning problems, particularly those problems involving dynamic environments and multiple agents. In each case, our primary emphasis will be on establishing a correspondence between actions in the real world and the formal notion of an action used in classical planning problems. Our goal, however, is not to understand the true nature of actions in the real world, but rather to determine how a particular mathematical construct may be used to model problems of interest.

Historically, the formal notion of an action developed in Section 2 was used to model problems that involve a single agent operating in an environment that remains essentially static, except while the agent is performing an action. To illustrate, imagine a robot in a closed room that contains a table on which a number of wooden blocks of equal size are stacked. The robot may be commanded (say, by radio) to move the blocks one at a time, either placing them elsewhere on the table or stacking them on top of other blocks. When issued a command, the robot will reach for the block to be moved, pick it up, position it over the desired location, and deposit it. Once the block has been deposited, the robot will stand motionless until the next command is received. Thus, except for those periods when the robot is carrying out a order, the robot and the blocks remain static (i.e., their respective positions remain fixed).

Imagine further that our goal is to have the robot arrange the blocks in a particular configuration. To determine an appropriate sequence of commands, all we need to know about each command are its net effects–that is, what the final respective positions of the robot and the blocks would be once the command has been carried out. The exact motion of the robot and the blocks during execution is unimportant. This leads us directly to the classical planning framework: First we can define a state to be a particular arrangement of the robot and blocks. Then we can catalogue the effects of a command by means of a set of current-state/next-state pairs specifying what the final disposition of the robot and blocks would be for each possible arrangement at the moment a command is invoked. This way of formalizing the problem ignores the details of the robot's motion while retaining all of the information relevant to planning a sequence of commands. It should be noted, however, that this formalization is possible only because the robot and the blocks remain stationary between the completion of one task and the start of the next.

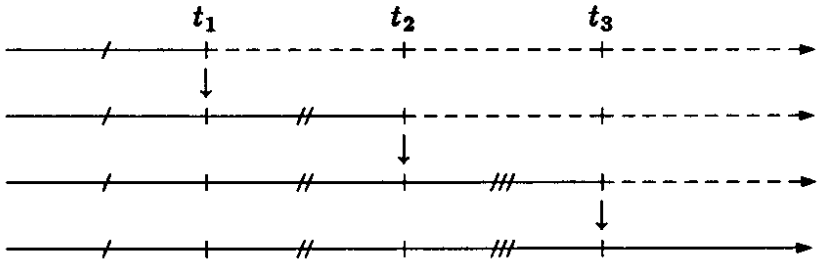
In many real-world situations, though, the world is not static but is in continuous state of flux. For example, suppose that our robot is trying to land a spacecraft on the moon. At any point in time, the robot may change the vehicle's flight path by adjusting the attitude of the vehicle and/or its thrust. Unlike the blocks in the previous example, however, the vehicle is in continuous motion until it lands (safely, we hope) on the lunar surface.

Single-agent, dynamic-world problems, such as the one just described, can be formulated in the classical planning framework by assigning new interpretations to the mathematical notions of state and state transition. In the static case, the world does not

change between the end of one action and the beginning of the next. This permits a state to represent everything that is true of the world at a given point in time between one action and the next. In a dynamic situation, however, this is not sufficient, since the world may be continually changing of its own accord from one moment to the next. Hence, what may be true at one point in time might not be true an instant later. How, then, is it possible to model actions in a dynamic environment as sets of current-state/next-state pairs? The key is to use states not for cataloguing what is true of the world at given instants in time, but rather for cataloguing what is true at each individual point in time, for all such points. To use McDermott's terminology [21, 23], a state according to this new interpretation is a *chronicle* of all that is true, was true, and will be true of the world, from the beginning of time through to the end of time.

Since we are interpreting states as chronicles, how should we interpret state transitions? The most sensible answer is to interpret a state transition as the initiation of an action or course of action at a particular point in time. Under this interpretation, when a state transition is made from one chronicle to the next, the past with respect to the new chronicle must be identical to the past with respect to the preceding chronicle. This reflects the commonsense intuition that the past is inaccessable and therefore cannot be changed. The future portion of the new chronicle, however, may be different. That portion must catalogue everything that would be true henceforth, *provided that no further actions are initiated.* In particular, it would record the events that take place as a result of executing the action whose initiation the state transition represents. If a new action is initiated, it must obviously follow all prior initiations. This corresponds to the commonsense intuition that time always moves forward, never backward.

This new way of interpreting states and state descriptions is illustrated in the figure below. States (i.e., chronicles) are shown as time lines, while state transitions are depicted as jumps from one time line to another that has the same past.



As an example, consider again the problem of landing a spacecraft on the moon, where the allowable actions are to adjust the attitude of the vehicle and the thrust. For a given adjustment, the future course of events will be dictated by the physics of the situation and will not change unless further adjustments are made. Each new adjustment places the spacecraft on a new trajectory, do that a new chronicle of events comes into force. Each new chronicle has the same past as the preceding one, since the previous history of the vehicle does not change; only the future is changed.

It is interesting to note that the ontology of time and action presented above is identical, within isomorphism, to the ontology put forth by McDermott [21, 23]. This isomorphism becomes evident from the fact that each of McDermott's forward-branching trees can be expanded into a set individual chronicles and a set of transitions from one chronicle to the next. Furthermore, a set of chronicles, together with a set of transitions between chronicles that share the same past, can be collapsed into a forward-branching tree. This isomorphism is significant, since it reinforces our earlier conjecture that it may be possible to transfer techniques for solving classical planning problems to frameworks that are better suited to the modeling of simultaneous actions and dynamic processes. A first step toward effecting this transfer is to establish such an isomorphism.

Thus far, we have considered problems involving a single agent acting in a world that may be static or dynamic. In some real-world situation, though, there may be several agents capable of performing tasks simultaneously. To maximize productivity, as many agents as possible should be working at any given time. Therefore, we would like to construct plans that coordinate the simultaneous execution of actions by these agents. In attempting to formulate such problems in the classical planning framework, though, we are faced with a major incongruity—i.e., that actions in the classical planning framework may be performed only sequentially whereas multiple agents tend to act simultaneously. In some situation, however, the effect of executing actions simultaneously is the same as doing them sequentially, regardless of the order of execution. This happens, for example, when the actions affect independent parts of the world and so do not interact. If such is the case, we can solve the problem by assuming that actions are to be carried out sequentially, and then transform the resulting sequential plan into a parallel plan for coordinating their simultaneous execution. This parallel plan could be in the form of a partial ordering of actions that specifies which actions may be performed simultaneously, which sequentially.

Such a partial order may be constructed in the following manner. First, to each action in the sequence we assign a unique symbol that identifies that action and its position in the sequence. Next, we construct a binary relation $\prec$ on these identifiers that will be used to define the partial order. The binary relation itself is defined as follows: We start by writting $x \prec y$ as shorthand for $\langle x, y \rangle \in \prec$. Then, for any pair of identifiers x and y, we define $x \prec y$ to be true if and only if the action corresponding to x precedes the action corresponding to y in the sequential plan and these two actions cannot be performed simultaneously. The desired partial order is then given by the *transitive closure* [27, 31] of $\prec$, which we write as $\prec^*$. Mathematically speaking, $\prec^*$ is the smallest binary relation such that

(1) If $x \prec y$ then $x \prec^* y$
(2) If $x \prec^* y$ and $y \prec^* z$ then $x \prec^* z$.

The relation $\prec^*$ specifies in the following way which actions may be performed simultaneously and which sequentially. If $x \prec^* y$, the action corresponding to x must be carried out before the action corresponding to y. On the other hand, if neither $x \prec^* y$ nor $y \prec^* x$, then the actions corresponding to x and y may be executed simultaneously.

It should be emphasized that the procedure just outlined works correctly if and only if the effect of performing actions simultaneously is the same as performing them sequentially. However, not all actions behave this way. Some actions have synergistic effects when performed simultaneously. As a result, their effects are different when performed simultaneously as compared with their sequential execution. For example, consider the effect when two agents lift opposite ends of a table upon which various objects have been placed. If the ends of the table are raised sequentially, the objects on the table might slide around and perhaps even fall off. On the other hand, if the ends are lifted simultaneously and with equal force, the objects on the table will remain more or less fixed. Chapman [3] gives a similar example involving the assembly of LEGO™ blocks. In his example, two agents are pressing down on opposite ends of a LEGO™ block that is resting on top of another LEGO™ block. If the agents apply equal force simultaneously, the first block will mate with the bottom block. On the other hand, if unequal force is applied or if equal force is applied sequentially, the first block will merely pivot and jam.

On the surface, it would appear that synergistic simultaneous actions cannot be dealt with in the classical planning framework, since the latter requires that actions be

performed sequentially. Nevertheless, by choosing the appropriate representation, some synergistic simultaneous actions can be made to look like sequential actions, thereby permitting their formulation in the classical planning framework. In the table illustration given above and in Chapman's LEGO™ example, this property can be achieved by modeling the forces that are acting upon on the objects. The overall motion of the objects is determined by the sum of the applied forces and the physical constraints. The primary effect of an action is to apply a force. Since the individual forces acting upon an object can be combined in any order to determine the net force, the actions of applying force can be considered sequentially. This allows the actions to be formulated as sets of current-state/next-state pairs, where each state is a chronicle of all that is true, was true, and will be true of the world, and where each state transition represents the effect of adding or subtracting a force vector at a particular instant. When a transition occurs, not only is the future portion of the chronicle modified, but a record is made noting the force that was added or deleted, along with the time at which this occurred. In this way, the net effect of simultaneously adding or subtracting forces can be determined through a sequence of state transitions.

This approach to determining the net effect of synergistic simultaneous actions can be generalized in the following way. The primary effect of an action is to establish a boundary condition at the point in time at which the action is initiated. The future course of events is then determined by these boundary conditions. In the table example and in Chapman's LEGO™ example, the boundary conditions are the forces applied to the objects. When actions are performed simultaneously, they add to and modify one another's boundary conditions. The resulting boundary conditions then determine the net effect of the actions. Furthermore, the boundary conditions is are modified in a way that enables the actions and their contributions to the boundary conditions to be considered sequentially and in any order. In this way, synergistic simultaneous actions can be made to resemble sequential actions, thus permitting the problem to be formulated in the classical planning framework.

A concrete example of this technique is presented in Section 5. First, however, suitable languages need to be established for representing states and actions.

4. DESCRIBING STATES AND ACTIONS

When solving complex problems, it is often impossible to deal with states and state transitions explicitly, since the number of possible states of the world may be infinite—or at least so large as to make it impractical to enumerate them all. In such cases, states and actions must be dealt with implicitly by creating languages that can express facts about states and actions. The problem would then be specifed and solved by using these languages. Facts in the languages would relate not just to individual states/actions, but to groups thereof. This is necessary so that an infinite number of states and state transitions can be described by a finite number of facts. For example, the lunar landing problem of Section 3 involves uncountably many states; nevertheless, only a page or two of mathematical formulas will suffice to define the problem precisely. Languages for describing states and actions can also be useful when solving problems that involve only a small number of states, as it may be more convenient to present the problems in terms of facts about states and actions than in terms of the states and actions themselves.

The principal requirement of languages for describing states and actions is unambiguousness. Descriptions are intended to represent facts, which are by definition not subject to interpretation. Hence, there can be no ambiguity as to whether or not a given state or action satisfies a given description. Therefore, a description must either be true with respect to a state or an action, or it must be false. Additionally, a description need not be complete: certain details might be left out, either because they are not known or because they are thought to be unimportant. Moreover, if a description is incomplete, there will be a multiplicity of states or actions that satisfy it.

Provided the above conditions are met, literally any language could be used to express facts about states and/or actions. For the purposes of this paper, two particular lanuages will be used. In keeping with much of the previous work in automatic planning, formulas of first-order logic [27, 34] will be used to express facts about states. To describe actions, a new language called ADL will be introduced. ADL has an advantage over its predecessors in that it can be used to define actions whose effects change depending on the circumstances in which they are performed. This feature is necessary to illustrate how multiagent, dynamic-world problems may be formulated in the classical planning framework.

ADL relies on a nonstandard semantics for first-order logic derived from the semantics of first-order dynamic logic [13]. It is also similar to the semantics of first-order logic as

presented by Manna and Waldinger [19]. Since a nonstandard semantics is being used, let us examine it in detail.

4.1 STATE DESCRIPTIONS AND FIRST-ORDER LOGIC

Since formulas shall be used to express facts about states, a given state will either satisfy a given formulas or it will not. To express this relationship symbolically, we shall write

$$s \models \varphi$$

to mean that state s satisfies the formula φ (or, equivalently, that φ is true when the world is in state s). Similarly,

$$s \not\models \varphi$$

means that state s does not satisfy the formula φ (or, equivalently, that φ is false when the world is in state s). One use of this notation is to define the set of initial states $\mathcal{I}$ and the set of goal states $\mathcal{G}$ for a planning problem in terms of state descriptions. Specifically, if Γ is a formula that describes the initial state of the world, and if G is a formula that describes the goals to be achieved, then

$$\mathcal{I} = \{s \mid s \models \Gamma\}$$
$$\mathcal{G} = \{s \mid s \models G\}$$

The concept in first-order logic that most closely satisfies the above relationship between states and formulas is that of an *algebraic structure*. An algebraic structure M is a tuple of the form

$$M = \langle D; r_1, \ldots, r_n; f_1, \ldots, f_m; d_1, \ldots, d_k \rangle .$$

where

(1) D is a nonempty set of objects called the *domain* of the structure
(2) $r_1, \ldots, r_n$ are relations on D
(3) $f_1, \ldots, f_m$ are functions on D
(4) $d_1, \ldots, d_k$ are distinguished elements of D.

As in set theory [31], relation r_i holds among elements $x_1, \ldots, x_n$ of D if and only if $\langle x_1, \ldots, x_n \rangle \in r_i$, and $f_i(x_1, \ldots, x_n) = y$ if and only if $\langle x_1, \ldots, x_n, y \rangle \in f_i$ (note that, in

set theory, a function is just a special kind of relation). Also, the functions that appear in an algebraic structure must be defined everywhere; that is, if f_i is a function of n variables, then, for every combination of n elements $x_1, \ldots, x_n$ of D, there must exist an element y of D such that $\langle x_1, \ldots, x_n, y \rangle \in f_i$. In this way, an algebraic structure completely specifies which relations and functions hold among the elements of D, and which do not. Borrowing the terminology of set theory, we shall call a function of n variables an n-ary function (i.e., *un*ary, *bin*ary, *tern*ary, etc.) and a relation among n elements an n-ary relation.

Algebraic structures are used to model a situation in the real world by choosing an appropriate set of objects for the domain of the structure, together with an appropriate collection of relations, functions, and distinguished elements for cataloguing the relevant information about the situation. For example, suppose we have a world that consists of a $\mathcal{TABLE}$ and three blocks $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$, where blocks $\mathcal{A}$ and $\mathcal{B}$ are resting on the $\mathcal{TABLE}$ and block $\mathcal{C}$ is stacked on top of block $\mathcal{A}$. This arrangement of blocks can be described by a single binary relation $\mathcal{ON}$ such that $\langle x, y \rangle \in \mathcal{ON}$ if and only if x is on top of y. Thus, we can use the algebraic structure BLK given below to represent this state of affairs:

$$BLK = \langle D_{BLK}; \mathcal{ON}_{BLK}; ; \mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{TABLE} \rangle \; .$$

where

$$D_{BLK} = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{TABLE}\}$$

and

$$\mathcal{ON}_{BLK} = \{\langle \mathcal{C}, \mathcal{A} \rangle, \langle \mathcal{A}, \mathcal{TABLE} \rangle, \langle \mathcal{B}, \mathcal{TABLE} \rangle\} \; .$$

In this algebraic structure, $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{TABLE}$ have all been identified as distinguished elements. The double semicolon indicates that there are no functions in the structure.

Formulas of a first-order language are used to describe facts about algebraic structures. In a first-order language, each relation, function, and distinguished element of an algebraic structure has a corresponding symbol in the language: the symbols corresponding to relations are called relation symbols, for functions they are called function symbols, and for distinguished elements they are called constant symbols. These symbols are commonly referred to as the *nonlogical symbols* and are defined by the user of the logic. For example, we might use the symbol On to stand for the $\mathcal{ON}$ relation discussed

above, and the symbols A, B, C, and $TABLE$ to stand for the distinguished elements $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{TABLE}$ respectively. For notational purposes, we shall write $M(Y)$ to mean the component of the algebraic structure M that corresponds to the nonlogical symbol Y. Thus, for the algebraic structure BLK defined above,

$$\begin{aligned} BLK(\text{On}) &= \{\langle \mathcal{C}, \mathcal{A}\rangle, \langle \mathcal{A}, \mathcal{TABLE}\rangle, \langle \mathcal{B}, \mathcal{TABLE}\rangle\} \\ BLK(A) &= \mathcal{A} \\ BLK(B) &= \mathcal{B} \\ BLK(C) &= \mathcal{C} \\ BLK(TABLE) &= \mathcal{TABLE} \end{aligned} \tag{4.1}$$

We shall often refer to $M(Y)$ as the *interpretation* of symbol Y in structure M.

In addition to the nonlogical symbols supplied by the user, a first-order language has a second group of symbols, called the *logical symbols*, that are built in. They are used to connect nonlogical symbols together to produce formulas. This group consists of variable symbols (e.g., $x_1, x_2, \ldots$) that range over objects in the domain of an algebraic structure, the symbols *TRUE* and *FALSE*, the equality symbol "$=$", the universal quantifier "$\forall$" (meaning *for all*), the existential quantifier "$\exists$" (meaning *there exists*), and the logical connectives "$\wedge$", "$\vee$", "$\neg$", "$\rightarrow$", and "$\leftrightarrow$" (corresponding in meaning, respectively, to *and*, *or*, *not*, *implies*, and *if and only if*).

To show how the logical symbols are used to construct formulas in a first-order language, we need to introduce the notion of a *term*. Terms are syntactic constructs that denote objects in the domain of an algebraic structure. For example, in Equation 4.1, the constant symbol A is a term that denotes the object $\mathcal{A}$ in the algebraic structure BLK. Terms are constructed from constant symbols, function symbols, and variables according to the following rules:

(1) Every constant symbol and every variable symbol is a term.

(2) If $t_1, \ldots, t_n$ are terms and F is an n-ary function symbol, then $F(t_1, \ldots, t_n)$ is a term.

(3) Nothing else is a term.

Using the notion of a term, the syntax for formulas of a first-order language is defined as follows:

(1) *TRUE* and *FALSE* are formulas.

(2) If t_1 and t_2 are terms, then $(t_1 = t_2)$ is a formula.

(3) If $t_1, \ldots, t_n$ are terms and R is an n-ary relation symbol, then $R(t_1, \ldots, t_n)$ is a formula.

(4) If φ and ψ are formulas, then $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \rightarrow \psi)$, and $(\varphi \leftrightarrow \psi)$ are formulas.

(5) If φ is a formula and x is a variable, then $\forall x\, \varphi$ and $\exists x\, \varphi$ are formulas.

(6) Nothing else is a formula.

To make formulas more readable, parentheses will be omitted whenever possible as long as this omission does not render the reading of the formula ambiguous. Also, formulas of the form $\neg(t_1 = t_2)$ will be written as $t_1 \neq t_2$ and sequences of quantifiers of the same type will be simplified by grouping the variables involved under one quantifier. For example, the formula

$$\forall x_1\, \forall x_2\, \forall x_3\, (((\mathrm{On}(x_1, x_3) \wedge \mathrm{On}(x_2, x_3)) \wedge \neg(x_3 = \mathit{TABLE})) \rightarrow (x_1 = x_2))$$

may be written as

$$\forall x_1\, x_2\, x_3\, ((\mathrm{On}(x_1, x_3) \wedge \mathrm{On}(x_2, x_3) \wedge x_3 \neq \mathit{TABLE}) \rightarrow x_1 = x_2)\,.$$

This particular formula is true if and only if all blocks are stacked so that at most one block is on top of another.

In the usual semantics of first-order logic, algebraic structures are used to assign truth-values to *closed formulas* (i.e., formulas in which all variables are associated with quantifiers). The formula given above is an example of a closed formula, since the variables x_1, x_2, and x_3 are all associated with the universal quantifier $\forall$. Given an algebraic structure, a closed formula will either be true or false with respect to the structure. The formula given above, for example, is true with respect to the algebraic structure BLK defined earlier. The relationship between algebraic structures and closed formulas is therefore identical to the relationship we desire between states and formulas. However, formulas that contain *free variables* (i.e., variables not associated with any quantifier)

do not necessarily have unique truth values with respect to algebraic structures. For example, the formula $\neg\mathrm{On}(x, A)$ would be true with respect to the algebraic structure BLK if x had the value $\mathcal{A}$, $\mathcal{B}$, or $\mathcal{TABLE}$; however, if x had the value $\mathcal{C}$, the formula would be false. What is usually done with free variables is to view them as being universally quantified. The formula $\neg\mathrm{On}(x, A)$ would then be equivalent to the formula $\forall x \neg\mathrm{On}(x, A)$.

When constructing plans, however, it is often convenient to use free variables as placeholders for terms yet to be chosen [1–6, 16, 25, 26, 28, 33, 35, 36, 38], or as parameters in a generalized plan [8]. Consequently, if the same free variable appears in more than one formula, we want that free variable to represent the same object in each of the formulas. This use, however, is inconsistent with the usual interpretation of free variables as being universally quantified. What is required is that free variables be assigned interpretations in a manner similar to constant symbols. This can be accomplished by defining states as follows:

Definition 4.2. A state is a pair of the form $\langle M, V\rangle$, where M is an algebraic structure and V is a function that maps variable symbols to elements of the domain of M.

The function V serves to assign interpretations to free variables. For notational convenience, we shall define $s(Y)$ to mean the interpretation of the symbol Y in state s; that is, if $s = \langle M, V\rangle$, then

$$s(Y) = \begin{cases} M(Y), & \text{if } Y \text{ is a relation, function, or constant symbol} \\ V(Y), & \text{if } Y \text{ is a variable symbol} \end{cases}$$

Also, we will refer to the domain of the algebraic structure of a state as the domain of the state.

Given a state s as defined above, and a formula φ, either φ will be true with respect to s or it will be false. To define this relation precisely, we first need to define a ***denotation function*** for terms. Terms refer to objects; hence, for a particular state, each term will denote a particular object in the domain of that state. Let $[\![\tau]\!]_s$ be the denotation of term τ in state s. Then $[\![\tau]\!]_s$ can be defined recursively as follows:

(1) $[\![C]\!]_s = s(C)$ for any constant symbol C

(2) $[\![x]\!]_s = s(x)$ for any variable x

(3) $[\![F(\tau_1,\ldots,\tau_n)]\!]_s = s(F)\,([\![\tau_1]\!]_s,\ldots,[\![\tau_n]\!]_s)$,

where F is a function symbol, $\tau_1,\ldots,\tau_n$ are terms, and $s(F)\,([\![\tau_1]\!]_s,\ldots,[\![\tau_n]\!]_s)$ is the result of applying the function denoted by F in state s to the denotations of the terms $\tau_1,\ldots,\tau_n$ in s.

To define the truth value of a quantified formula, an additional bit of notation is required. We shall write $s[x \mapsto d]$ to mean the state obtained by changing the interpretation of variable x to d in state s. Thus, the interpretation of a symbol Y in $s[x \mapsto d]$ is given by

$$s[x \mapsto d](Y) = \begin{cases} s(Y) & \text{if } Y \text{ is not the variable } x \\ d & \text{if } Y \text{ is the variable } x \end{cases}$$

The above notation permits the satisfaction relation $s \models \varphi$ between a state s and a formula φ to be defined recursively as follows:

(1) $s \models \mathit{TRUE}$

(2) $s \not\models \mathit{FALSE}$

(3) $s \models (\tau_1 = \tau_2)$ if and only if $[\![\tau_1]\!]_s = [\![\tau_2]\!]_s$

(4) $s \models R(\tau_1,\ldots,\tau_n)$ if and only if $\langle [\![\tau_1]\!]_s,\ldots,[\![\tau_n]\!]_s\rangle \in s(R)$

(5) $s \models (\varphi \wedge \psi)$ if and only if $s \models \varphi$ and $s \models \psi$

(6) $s \models (\varphi \vee \psi)$ if and only if $s \models \varphi$ or $s \models \psi$, or both

(7) $s \models \neg\varphi$ if and only if $s \not\models \varphi$

(8) $s \models (\varphi \rightarrow \psi)$ if and only if $s \not\models \varphi$ or $s \models \psi$, or both

(9) $s \models (\varphi \leftrightarrow \psi)$ if and only if $s \models \varphi$ and $s \models \psi$, or $s \not\models \varphi$ and $s \not\models \psi$

(10) $s \models \forall x\,\varphi$ if and only if $s[x \mapsto d] \models \varphi$ for every element d in the domain of s

(11) $s \models \exists x\,\varphi$ if and only if $s[x \mapsto d] \models \varphi$ for some d in the domain of s.

4.2 ADL: A LANGUAGE FOR DESCRIBING ACTIONS

Let us now consider a language for describing actions. This language, called ADL for Action Description Language, has the interesting property that it is syntactically similar to the STRIPS operator language of Fikes and Nilsson [7], yet it has relatively the same expressive power as the situation calculus of McCarthy and Hayes [20]. In fact, any classical planning problem that can be formulated in the situation calculus can also be formulated with ADL, and vice versa (a formal proof of the equivalence appears elsewhere [25], but is beyond the scope of the current paper). As a result, ADL can be used to describe actions whose effects are highly dependent upon the state of the world in which the action is performed. However, the syntactic properties of ADL enable the frame problem of the situation calculus to be circumvented—at least to the extent that one has only to specify the changes that take place when an action is performed, not what remains the same. The latter is provided implicitly by the language. Consequently, ADL has an advantage over other languages in that it is powerful yet convenient to use. The version of ADL presented here is an extension of a similar language introduced in an earlier technical report [24].

ADL is intended to be a practical language for describing actions. To arrive at this language, certain constraints were imposed on the kinds of actions that could be described in ADL. The first constraint is that the set of states in which an action can be performed must be representable as a well-formed formula. In other words, for an action a to be describable in ADL, there must exist a formula π^a such that

$$\mathrm{dom}(a) = \{s \mid s \models \pi^a\} .$$

This formula is called the *precondition* of the action. Note that this is a standard requirement incorporated into all planning systems to date.

The second constraint is that an action should not alter the domain of a state. That is, for any action a that can be described in ADL, and for any pair of states $\langle s, t\rangle \in a$, the domains of s and t must be identical. This requirement is of concern only when we wish to describe actions that create or destroy objects in the world. An example of such an action would be the GENSYM function in LISP, which creates new LISP atoms. Because actions are required to preserve the domain of a state, the effect of creating or destroying objects must be simulated by introducing a unary relation, say U, where $U(x)$ is true if

and only if object x "actually" exists. Objects would then be "created" or "destroyed" by modifying the interpretation of U. Of course, this would require that the domain of a state include all objects that could possibly exist. Note that this is precisely the way in which GENSYM is implemented in a real computer: GENSYM does not create LISP atoms "out of thin air," but rather locates an area of unused memory and claims it for use as a new atom.

The third constraint is that an action must preserve the interpretations of free variables; that is, for any action a that can be described in ADL, and for any pair of states $\langle s, t\rangle \in a$, it must be the case that $s(x) = t(x)$ for every variable symbol x. The reason for this is that we shall want to use free variables as parameters in a plan. Consequently, a free variable that appears at more than one point in a plan should represent the same object at each of those points. Hence, we must require that actions preserve the interpretations of free variables.

The fourth constraint is that actions described in ADL be deterministic; that is, for any state $s \in \mathrm{dom}(a)$, there is a unique state t such that $\langle s, t\rangle \in a$. The rationale for this constraint is that it allows an action to be decomposed into a collection of functions—one for each relation symbol, function symbol, and constant symbol. Each function in the collection is a mapping from states to interpretations of the corresponding symbol in the succedent states. Thus, if f_Y is the function corresponding to symbol Y for action a, and if $\langle s, t\rangle \in a$, then $t(Y) = f_Y(s)$.

To provide a way of specifying these functions, we shall introduce our fifth and final constraint, which is that each function be representable as a formula. That is, each function f_Y corresponding to a symbol Y is then defined by a formula φ_Y, such that

(1) For each n-ary relation symbol R, $R(x_1, \ldots, x_n)$ is true in the succedent state if and only if $\varphi_R(x_1, \ldots, x_n)$ was true previously (where $x_1, \ldots, x_n$ are the free variables of φ_R).

(2) For each n-ary function symbol F, $F(x_1, \ldots, x_n) = w$ is true in the succedent state if and only if $\varphi_F(x_1, \ldots, x_n, w)$ was true previously.

(3) For each constant symbol C, $C = w$ is true in the succedent state if and only if $\varphi_C(w)$ was true previously.

Stated mathematically, for every pair of states $\langle s, t\rangle \in a$, and for every n-ary relation symbol R, every n-ary function symbol F, and every constant symbol C, there must exist formulas $\varphi_R(x_1, \ldots, x_n)$, $\varphi_F(x_1, \ldots, x_n, w)$, and $\varphi_C(w)$, such that

(1) $t(R) = \{\langle d_1, \ldots, d_n\rangle \mid s[x_i \mapsto d_i] \models \varphi_R(x_1, \ldots, x_n)\}$

(2) $t(F) = \{\langle d_1, \ldots, d_n, e\rangle \mid s[x_i \mapsto d_i][w \mapsto e] \models \varphi_F(x_1, \ldots, x_n, w)\}$

(3) $t(C) = d$ such that $s[w \mapsto d] \models \varphi_C(w)$.

For example, suppose we have an action for placing block B on top of block C. After this action is applied, block B is situated atop block C and every block except B remains where it was. Therefore, $\mathrm{On}(x, y)$ is true following the action if and only if

$$(x = B \wedge y = C) \vee (x \neq B \wedge \mathrm{On}(x, y))$$

was true just prior to execution. In other words, if the action is performed in state s, the interpretation of On in the succedent state is the set of ordered pairs $\langle d, e\rangle$, such that

$$s[x \mapsto d][y \mapsto e] \models (x = B \wedge y = C) \vee (x \neq B \wedge \mathrm{On}(x, y)) \ .$$

If this action were performed with the blocks stacked as described in Section 4.1, where the interpretation of On was $\{\langle \mathcal{B}, \mathcal{TABLE}\rangle, \langle \mathcal{C}, \mathcal{A}\rangle, \langle \mathcal{A}, \mathcal{TABLE}\rangle\}$, then the resulting interpretation of On would be $\{\langle \mathcal{B}, \mathcal{C}\rangle, \langle \mathcal{C}, \mathcal{A}\rangle, \langle \mathcal{A}, \mathcal{TABLE}\rangle\}$.

When solving a planning problem, it is important to know exactly what modifications an action induces in a state before the appropriate actions for achieving the intended goals can be selected. Therefore, we shall express the above-defined φ_R's, φ_F's and φ_C's in terms of other formulas that make the modifications explicit, and then deal exclusively with these other formulas. For relation symbols, this means expressing each φ_R associated with an action a in terms of two other formulas, α_R and δ_R, which, respectively, describe the additions to and the deletions from the interpretation of R. In other words, if action a is performed in state s, the interpretation of R in the succedent state is given by

$$t(R) = (s(R) - D_R) \cup A_R \ ,$$

where $s(R)$ is the interpretation of R in state s, D_R the set of tuples to be deleted from $s(R)$, and A_R the set of tuples to be added to $s(R)$, and where A_R and D_R are given by

$$A_R = \{\langle d_1, \ldots, d_n\rangle \mid s[x_i \mapsto d_i] \models \alpha_R(x_1, \ldots, x_n)\}$$
$$D_R = \{\langle d_1, \ldots, d_n\rangle \mid s[x_i \mapsto d_i] \models \delta_R(x_1, \ldots, x_n)\}\,.$$

We shall further require that A_R and D_R be nonoverlapping (i.e., $A_R \cap D_R = \emptyset$), so that additions and deletions may be done in any order. Consequently,

$$(s(R) - D_R) \cup A_R = (s(R) \cup A_R) - D_R$$

This requires that, for every state $s \in \mathrm{dom}(a)$,

$$s \models \neg\exists x_1 \cdots x_n\,(\alpha_R(x_1, \ldots, x_n) \wedge \delta_R(x_1, \ldots, x_n))$$

Given formulas α_R and δ_R, as defined above, formula φ_R is expressed by

$$\alpha_R(x_1, \ldots, x_n) \vee (\neg\delta_R(x_1, \ldots, x_n)) \wedge R(x_1, \ldots, x_n) \tag{4.3}$$

To paraphrase, $R(x_1, \ldots, x_n)$ is true after performing action a if and only if action a made it true, or it was true beforehand and action a did not make it false. Note that it is possible to find appropriate formulas α_R and δ_R for any given formula φ_R; for example, we can let $\alpha_R(x_1, \ldots, x_n)$ be the formula $\varphi_R(x_1, \ldots, x_n)$ and $\delta_R(x_1, \ldots, x_n)$ be $\neg\varphi_R(x_1, \ldots, x_n)$. For efficient problem solving, though, α_R and δ_R should be chosen to reflect the minimal modifications required in the interpretation of R. For example, for the block-stacking action described previously, a suitable $\alpha_{\mathrm{On}}(x, y)$ would be $(x = B \wedge y = C)$ and a suitable $\delta_{\mathrm{On}}(x, y)$ would be $(x = B \wedge y \neq C)$. Note that $\delta_{\mathrm{On}}(x, y)$ cannot be $(x = B)$, since $\alpha_{\mathrm{On}}(x, y)$ and $\delta_{\mathrm{On}}(x, y)$ are constrained from being true simultaneously.

The formulas defining the interpretations of the function symbols in the succedent state can be restructured in much the same way as the formulas for relation symbols. In the case of functions, however, we can take advantage of the fact that a function must be defined everywhere, as required by the definition of an algebraic structure. Consequently, $F(x_1, \ldots, x_n) = w$ is true after an action has been performed if and only if the action changed the value of $F(x_1, \ldots, x_n)$ to w, or the action preserved the value of $F(x_1, \ldots, x_n)$ and $F(x_1, \ldots, x_n) = w$ was true previously. These modifications can be described by a single formula $\mu_F(x_1, \ldots, x_n, w)$ that is true if and only if the value of $F(x_1, \ldots, x_n)$ is to be changed to w when the action is applied. Since functions have unique values, μ_F

must have the property that, for any instantiation of $x_1, \ldots, x_n$, either there is a unique w for which $\mu_F(x_1, \ldots, x_n, w)$ is true or there are no w's for which $\mu_F(x_1, \ldots, x_n, w)$ is true. Given such a μ_F, the formula φ_F defining the interpretation of F in the succedent state is expressed by

$$\mu_F(x_1, \ldots, x_n, w) \vee (\neg \exists w\, [\mu_F(x_1, \ldots, x_n, w)] \wedge F(x_1, \ldots x_n) = w) \ . \qquad (4.4a)$$

As with α_R and δ_R, an appropriate μ_F can be found given any formula φ_F; for example, we can let $\mu_F(x_1, \ldots, x_n, w)$ be $\varphi_F(x_1, \ldots, x_n, w)$. However, for efficient problem solving, μ_F should be chosen to reflect the minimal modifications required in the interpretation of F. As an example, suppose that we wish to model the assignment statement $U \leftarrow V$, where U and V are program variables. To do so, we could have a function Val that maps program variables to their values, plus an action that updates $\mathrm{Val}(U)$ to be the value of $\mathrm{Val}(V)$. An appropriate update condition $\mu_{\mathrm{Val}}(x, w)$ for this action would then be $(x = U \wedge w = \mathrm{Val}(V))$.

Constant symbols are handled in exactly the same way as function symbols, since the former may be thought of as functions without arguments. Therefore, φ_C is given by

$$\mu_C(w) \vee (\neg \exists w\, [\mu_C(w)] \wedge C = w) \qquad (4.4b)$$

Note that Formula (4.4*b*) is simply a special case of Formula (4.4*a*) in which $n = 0$.

The syntax of ADL is designed to provide a convenient means of specifying the α, δ, μ, and π formulas that define an action. To illustrate the syntax of ADL, consider the following two actions described in ADL:

Put(p, q)
  PRECOND: $p \neq q$, $p \neq TABLE$, $\forall z \neg \mathrm{On}(z, p)$,
    $[q = TABLE \vee \forall z \neg \mathrm{On}(z, q)]$
  ADD: $\mathrm{On}(p, q)$
  DELETE: $\mathrm{On}(p, z)$ for all z such that $z \neq q$

Assign(u, v)
  UPDATE: $\mathrm{Val}(u) \leftarrow \mathrm{Val}(v)$

The first action, $\mathrm{Put}(p,q)$, is a block-stacking action for placing a block p on top of an object q, where q may be either another block or the table. The second action, $\mathrm{Assign}(u,v)$, simulates a statement that might appear in a programming language for assigning to variable u the value of variable v.

As the examples illustrate, a description in ADL consists of a name, an optional parameter list, and four optional groups of clauses labeled PRECOND, ADD, DELETE, and UPDATE. The PRECOND group consists of a list of formulas that define the set of states in which an action may be performed. Every formula in the list must be true when the action is performed; hence, the precondition π for the action is the conjunction of these formulas. If the list is empty, π is taken to be the formula *TRUE*, meaning that the action may be performed in every state. Thus, the precondition of $\mathrm{Assign}(u,v)$ is the formula *TRUE*, whereas the precondition of $\mathrm{Put}(p,q)$ is the formula

$$p \neq q \wedge p \neq \mathit{TABLE} \wedge \forall z \neg \mathrm{On}(z,p) \wedge [q = \mathit{TABLE} \vee \forall z \neg \mathrm{On}(z,q)]$$

The α and δ formulas for an action are specified by the ADD and DELETE groups, respectively. Each group consists of a set of clauses of the following forms:

(1) $R(t_1,\ldots,t_n)$
(2) $R(t_1,\ldots,t_n)$ if ψ
(3) $R(t_1,\ldots,t_n)$ for all $z_1,\ldots,z_k$
(4) $R(t_1,\ldots,t_n)$ for all $z_1,\ldots,z_k$ such that ψ ,

where R is a relation symbol, $t_1,\ldots,t_n$ are terms, ψ is a formula, and $z_1,\ldots,z_k$ are variable symbols that appear in terms $t_1,\ldots,t_n$ but not in the parameter list. Each of the clauses in an ADD group corresponds to an *add condition*, $\hat{\alpha}_R$, for some relation symbol R. The formula α_R that defines the set of tuples to be added to the interpretation of R is obtained by taking the conjunction of each of the add conditions for R defined in the group. If no add conditions are specified for symbol R, α_R is taken to be the formula *FALSE*, meaning that no tuples are to be added to the interpretation of R. The semantics of the DELETE group is similar to the ADD group except that, in this case, each clause corresponds to a *delete condition*, $\hat{\delta}_R$, for some relation symbol R. The conjunction of the delete conditions for R defines the formula δ_R. If no delete conditions are specified for R, δ_R is taken to be the formula *FALSE*, meaning that no tuples are to

be deleted from the interpretation of R. The add/delete conditions that correspond to each of the four types of clauses are listed in the table below:

Clause	$\hat{\alpha}_R(x_1,\ldots,x_n)/\hat{\delta}_R(x_1,\ldots,x_n)$
$R(t_1,\ldots,t_n)$	$(x_1 = t_1 \wedge \cdots \wedge x_n = t_n)$
$R(t_1,\ldots,t_n)$ if ψ	$(x_1 = t_1 \wedge \cdots \wedge x_n = t_n \wedge \psi)$
$R(t_1,\ldots,t_n)$ for all $z_1,\ldots,z_k$	$\exists z_1 \cdots z_n\,(x_1 = t_1 \wedge \cdots \wedge x_n = t_n)$
$R(t_1,\ldots,t_n)$ for all $z_1,\ldots,z_k$ such that ψ	$\exists z_1 \cdots z_n\,(x_1 = t_1 \wedge \cdots \wedge x_n = t_n \wedge \psi)$

Thus, $\alpha_{\mathrm{On}}(x,y)$ for $\mathrm{Put}(p,q)$ is given by

$$\alpha_{\mathrm{On}}(x,y) \equiv (x = p \wedge y = q)\,.$$

Similarily, $\delta_{\mathrm{On}}(x,y)$ is given by

$$\delta_{\mathrm{On}}(x,y) \equiv \exists z\,(x = p \wedge y = z \wedge z \neq q)\,,$$

which simplifies to

$$\delta_{\mathrm{On}}(x,y) \equiv (x = p \wedge y \neq q)\,.$$

In the above equations, the symbol "$\equiv$" is used to denote equality between formulas. A separate symbol is used so as to avoid possible confusion with the logical symbol "$=$".

The UPDATE group is used to specify the μ-formulas that define the interpretations of the function symbols in the succedent state. The UPDATE group consists of a set of clauses of the following forms:

(1) $C \leftarrow t$
(2) $C \leftarrow t$ if ψ
(3) $C \leftarrow t$ for all $z_1,\ldots,z_k$
(4) $C \leftarrow t$ for all $z_1,\ldots,z_k$ such that ψ
(5) $F(t_1,\ldots,t_n) \leftarrow t$
(6) $F(t_1,\ldots,t_n) \leftarrow t$ if ψ
(7) $F(t_1,\ldots,t_n) \leftarrow t$ for all $z_1,\ldots,z_k$
(8) $F(t_1,\ldots,t_n) \leftarrow t$ for all $z_1,\ldots,z_k$ such that ψ ,

where C is a constant symbol, F is a function symbol, $t_1,\ldots,t_n,t$ are terms, ψ is a formula, and $z_1,\ldots,z_k$ are variable symbols that appear in terms $t_1,\ldots,t_n,t$ but not in the parameter list. Each clause in an UPDATE group corresponds to an *update condition*, $\hat{\mu}_F$, for some function symbol F, or to an update condition, $\hat{\mu}_C$, for some constant symbol C. The formula μ_F that defines the modifications in the interpretation of F is obtained by taking the conjunction of each of the update conditions $\hat{\mu}_F$ defined in the group. Likewise for the formula μ_C. If no update conditions are specified for a function/constant symbol F/C, μ_F/μ_C is taken to be the formula *FALSE*, meaning that no modifications are made in the interpretation of F/C. The update conditions that correspond to each of the eight types of clauses are listed below:

Clause	$\hat{\mu}_C(w)/\hat{\mu}_F(x_1,\ldots,x_n,w)$
$C \leftarrow t$	$(w = t)$
$C \leftarrow t$ if ψ	$(w = t \wedge \psi)$
$C \leftarrow t$ for all $z_1,\ldots,z_k$	$\exists z_1,\ldots,z_k\,(w = t)$
$C \leftarrow t$ for all $z_1,\ldots,z_k$ such that ψ	$\exists z_1,\ldots,z_k\,(w = t \wedge \psi)$
$F(t_1,\ldots,t_n) \leftarrow t$	$(x_1 = t_1 \wedge \cdots \wedge x_n = t_n \wedge w = t)$
$F(t_1,\ldots,t_n) \leftarrow t$ if ψ	$(x_1 = t_1 \wedge \cdots \wedge x_n = t_n \wedge w = t \wedge \psi)$
$F(t_1,\ldots,t_n) \leftarrow t$ for all $z_1,\ldots,z_k$	$\exists z_1 \cdots z_n\,(x_1 = t_1 \wedge \cdots \wedge x_n = t_n \wedge w = t)$
$F(t_1,\ldots,t_n) \leftarrow t$ for all $z_1,\ldots,z_k$ such that ψ	$\exists z_1 \cdots z_n\,(x_1 = t_1 \wedge \cdots \wedge x_n = t_n \wedge w = t \wedge \psi)$

For example, for the Put(p,q) action defined earlier,

$$\mu_A(w) \equiv \mathit{FALSE} \qquad \mu_C(w) \equiv \mathit{FALSE}$$
$$\mu_B(w) \equiv \mathit{FALSE} \qquad \mu_{TABLE}(w) \equiv \mathit{FALSE}$$

and, for the Assign(u,v) action,

$$\mu_{\mathrm{Val}}(x,w) \;\equiv\; (x = u \wedge w = \mathrm{Val}(v))\,.$$

As the forgoing discussion demonstrates, ADL has a well-defined semantics. Actions are defined by providing the appropriate add, delete, and update conditions for the appropriate symbols. As long as these formulas obey the constraints imposed earlier, the

descriptions in ADL will define a unique set of current-state/next-state pairs. To summarize these constraints, $\alpha_R^a(x_1, \ldots, x_n)$ and $\delta_R^a(x_1, \ldots, x_n)$ must be mutually exclusive for every n-ary relation symbol R, and $\mu_F^a(x_1, \ldots, x_n, w)$ must define a partial function mapping $x_1, \ldots, x_n$ to w for every n-ary function symbol (constant symbol) F. Because actions are described directly in terms of their effects on the state of the world, and not in terms of the formulas they make true/false, ADL avoids the semantic pitfalls of the STRIPS operator language that are discussed by Lifschitz [18].

To reason about what formulas are made true/false by actions described in ADL, *regression operators* [38] are first constructed for these actions. A regression operator for an action described in ADL is a function mapping formulas to formulas that defines the necessary and sufficient conditions that must hold before an action is performed for some condition to be true afterward. Regression operators enable us to determine whether some formula φ holds at a point p in a plan by providing us with a formula ψ that would have to be true in the initial state for φ to be true at point p. The formula ψ can then be compared with the initial state to ascertain the truth of φ at point p. The construction of regression operators from ADL descriptions and their use in plan synthesis are described elsewhere [24, 25].

5. EXAMPLES

In this section, we shall present a number of examples illustrating how multiagent, dynamic-world problems can be formulated and solved as classical planning problems. Let us first consider a specific multiagent, static-world problem. Suppose we have a world consisting of four blocks, A, B, C and D, stacked on a table, A initially on top of B, C atop D, and both B and D resting on the table. Let us suppose further that our goal is to have A on top of D and C on top of B. However, instead of having only a single robot to control, we now have two robots, R_1 and R_2. They may be commanded to both pick up and deposit blocks. But these particular robots are unable to carry more than one block at a time. Their actions can be described in ADL as follows:

PickUp (rbt, blk)

PRECOND: $\text{Robot}(rbt)$, $blk \neq TABLE$, $\forall z \neg \text{Holding}(rbt, z)$, $\forall z \neg \text{Holding}(z, blk)$, $\forall z \neg \text{On}(z, blk)$

ADD: $\text{Holding}(rbt, blk)$

DELETE: $\text{On}(blk, z)$ for all z

PutDown (rbt, obj)

PRECOND: $\text{Robot}(rbt)$, $obj = TABLE \vee \forall z \neg \text{On}(z, obj)$

ADD: $\text{On}(z, obj)$ for all z such that $\text{Holding}(rbt, z)$

DELETE: $\text{Holding}(rbt, z)$ for all z

Both actions require the parameter *rbt* to designate one of the robot, R_1 or R_2. The PickUp action corresponds to the command that causes robot *rbt* to pick up object *blk*. For the robot to do so, *blk* cannot be the table, the robot cannot be holding any other object, no other robot can be holding *blk*, and nothing can be on top of *blk*. After robot *rbt* performs this action, it will be holding block *blk*, and block *blk* will be removed from its prior resting place. The PutDown action corresponds to the command that causes robot *rbt* to put whatever it is holding on top of object *obj*. To do so, either *obj* must be the table or there must be nothing on top of *obj*. When this action is executed, all objects held by robot *rbt* are placed on top of *obj*, with *rbt* releasing its hold on these objects.

Using the planning technique described in an earlier paper [24], we can obtain the following plan for placing block A atop block D, and block C atop block B:

$$\text{PickUp}(R_1, A) \rightarrow \text{PickUp}(R_2, C) \rightarrow \text{PutDown}(R_1, D) \rightarrow \text{PutDown}(R_2, B)$$

To construct a parallel plan, we note that $\text{PickUp}(R_1, A)$ and $\text{PickUp}(R_2, C)$ can be performed simultaneously by the robots, as can $\text{PutDown}(R_1, D)$ and $\text{PutDown}(R_2, B)$. However, neither $\text{PickUp}(R_1, A)$ nor $\text{PickUp}(R_2, C)$ can be executed at the same time

as PutDown (R_1, D) or PutDown (R_2, B). Thus, the parallel plan would be to allow PickUp (R_1, A) and PickUp (R_2, C) to be performed simultaneously, after which actions PutDown (R_1, D) and PutDown (R_2, B) could also be concurrent.

Let us now consider a single-agent, dynamic-world problem: specifically, the lunar-landing problem described in Section 3. In this example, the thrust and attitude of the vehicle are adjusted simultaneously by a single robot. We shall assume a two-dimensional version of the problem in which the controls can be set at any point in time to provide a desired acceleration at a desired angular orientation. Once set, the acceleration and orientation are maintained until the next adjustment. If we disregard the possibility of running out of fuel or crashing into the moon, the effect of setting the controls can be formulated in ADL as follows:

Adjust(a, θ, t')

PRECOND: $t_0 \leq t', \ min \leq a \leq max$

UPDATE: $a_0 \leftarrow a, \ \theta_0 \leftarrow \theta, \ t_0 \leftarrow t'$

$x(t) \leftarrow \frac{1}{2} a \sin\theta \, (t - t')^2 + v_x(t')(t - t') + x(t')$ fa t st $t \geq t'$

$y(t) \leftarrow \frac{1}{2}(g - a\cos\theta)(t - t')^2 + v_y(t')(t - t') + x(t')$ fa t st $t \geq t'$

$v_x(t) \leftarrow a \sin\theta \, (t - t') + v_x(t')$ fa t st $t \geq t'$

$v_y(t) \leftarrow (g - a\cos\theta)(t - t') + v_y(t')$ fa t st $t \geq t'$

where a is the net acceleration, θ the angular orientation, and t' the time at which the controls are adjusted, and where "fa" is shorthand for "for all", and "st" shorthand for "such that". The constants a_0, θ_0, and t_0 denote the acceleration, orientation, and time of the most recent adjustment, respectively. The functions $x(t)$ and $y(t)$ define the x and y coordinates of the vehicle as functions of time, while $v_x(t)$ and $v_y(t)$ define the component velocities of the vehicle along the x and y axes. Since t_0 is the time of the most recent adjustment, Adjust(a, θ, t') has the precondition that t' be greater than or equal to t_0. Adjust(a, θ, t') also has the precondition that the net acceleration lie within certain bounds, reflecting the limitations of the vehicle's engine. When an adjustment is made, the future is thereby altered, placing the vehicle on a new trajectory. The history of the vehicle's travel, however, is preserved.

The goal of a lunar-landing problem is to land safely within a certain distance of some targeted point. To ensure a safe landing, the x and y velocities must be sufficiently small and the attitude of the vehicle must be sufficiently close to vertical. This goal can be expressed as follows:

$$\exists t\,[\; y(t) = 0 \wedge\; -d_x \leq\; x(t)\; \leq d_x\; \wedge\quad 0 \leq v_y(t) \leq d_{v_y}$$
$$\wedge -d_{v_x} \leq v_x(t) \leq d_{v_x} \wedge -d_\theta \leq\; \theta_0\; \leq d_\theta\;]$$

In the initial state, t_0 would represent the time at which control of the vehicle is turned over to the planner, a_0 would represent the net acceleration at that time, and θ_0 would represent the angular orientation. The functions $x(t)$, $y(t)$, $v_x(t)$, and $v_y(t)$ would have to be defined accordingly, and would have to reflect the position and velocity of the craft at time t_0. In addition, the arithmetic and trigonometric functions would have to be defined. With the appropriate formulas selected for initial-state description, the problem could then be solved by using the techniques presented elsewhere [25].

To give an example of a multiagent, dynamic-world problem, let us again consider a lunar-landing situation, but this time with the thrust and attitude adjusted independently by different robots. Although this is not the most representative of multiagent problems, it does serve to illustrate how the concept of boundary conditions can be used to handle the problem of synergistic simultaneous actions. As discussed in Section 3, by applying this notion, synergistic simultaneous actions can be made to resemble sequential actions. In the case of the lunar-landing problem, we can use acceleration and angular orientation as boundary conditions. This permits us to define the actions for setting the thrust and attitude as follows:

Thrust(a, t')

PRECOND: $t_0 \leq t',\ \ min \leq a \leq max$

UPDATE: $a_0 \leftarrow a,\ \ t_0 \leftarrow t'$

$x(t) \leftarrow \frac{1}{2} a \sin\theta_0\,(t-t')^2 + v_x(t')(t-t') + x(t')$ fa t st $t \geq t'$

$y(t) \leftarrow \frac{1}{2}(g - a\cos\theta_0)(t-t')^2 + v_y(t')(t-t') + x(t')$ fa t st $t \geq t'$

$v_x(t) \leftarrow a\sin\theta_0\,(t-t') + v_x(t')$ fa t st $t \geq t'$

$v_y(t) \leftarrow (g - a\cos\theta_0)(t-t') + v_y(t')$ fa t st $t \geq t'$

Angle(θ, t')

PRECOND: $t_0 \leq t'$

UPDATE: $\theta_0 \leftarrow \theta,\ t_0 \leftarrow t'$

$$x(t) \leftarrow \tfrac{1}{2}a_0 \sin\theta\,(t-t')^2 + v_x(t')(t-t') + x(t') \ \text{ fa } t \text{ st } t \geq t'$$

$$y(t) \leftarrow \tfrac{1}{2}(g - a_0\cos\theta)(t-t')^2 + v_y(t')(t-t') + x(t') \ \text{ fa } t \text{ st } t \geq t'$$

$$v_x(t) \leftarrow a_0 \sin\theta\,(t-t') + v_x(t') \ \text{ fa } t \text{ st } t \geq t'$$

$$v_y(t) \leftarrow (g - a_0\cos\theta)(t-t') + v_y(t') \ \text{ fa } t \text{ st } t \geq t'$$

Adjusting the thrust does not affect the attitude of the vehicle; hence, the effect of this action is to update the acceleration boundary condition and thus to adjust the vehicle's future trajectory in accordance with the new acceleration and the previous angular orientation. Similarly, adjusting the attitude does not affect the thrust; hence, the effect of this action is to update the angular-orientation boundary condition and to adjust the vehicle's future trajectory in accordance with the new angular orientation and the previous acceleration. By defining the above actions in this manner, we obtain the desired property that performing Thrust(a_1, t_1) followed by Angle(θ_1, t_1) has the same effect as performing Angle(θ_1, t_1) followed by Thrust(a_1, t_1). Furthermore, their combined effect is the same effect as Adjust(a_1, θ_1, t_1). Thus, by using acceleration and angular orientation as boundary conditions, we have effectively made synergistic simultaneous actions resemble sequential actions. A problem involving these actions can therefore be solved by first constructing a sequential plan and then transforming it into a parallel plan as described in Section 3.

Technically speaking, the position and velocity of the vehicle at the time an adjustment is made should also be considered as boundary conditions, since the future motion of the vehicle depends on them. However, it is the thrust and attitude that are being modified directly by the actions; the position and velocity at the time of adjustment are not modified, but instead serve as continuity constraints. This suggests that, when the concept of boundary conditions is applied, it is important to distinguish between those conditions that are modified by an action directly and those that play the role of continuity constraints.

6. SUMMARY

We have shown how some multiagent, dynamic-world problems may be formulated in the classical planning framework, thereby enabling them to be solved using classical planning techniques. The two main obstacles to formulating such problems this way is representing simultaneous actions and representing continuous processes. As was explained, continuous processes can be represented in the classical planning framework by interpreting (1) a state as a chronicle of all that is true, was true, and will be true of the world at every point in time, and (2) a state transition as the initiation of an action or course of action. It was also shown that the classical framework is adequate for handling some simultaneous actions in spite of the fact that actions in that must be performed sequentially. By representing a problem in the appropriate manner, some simultaneous actions can be made to look like sequential ones, permitting their formulation in the classical planning framework. A general approach to representing actions, based on the concept of boundary conditions, was introduced to accomplish this objective. Whether this approach can be applied in all cases is left to subsequent research. In any event, the fact that some simultaneous actions can be represented in the classical planning framework suggests the feasibility of transferring techniques for solving classical planning problems to other frameworks that are specifically designed for modeling simultaneous actions and dynamic processes.

As part of the presentation, ADL, a new language for describing actions and their effects, was introduced. Because it combines the best features of the STRIPS operator language [7] and the situation calculus [20], ADL offers the advantage of both expressive power and notational convenience. Furthermore, ADL avoids the semantic pitfalls of the STRIPS operator language that are discussed by Lifschitz [18]. This is done by describing the effects of actions directly in terms of state transformations, as opposed to transformations on descriptions of states.

Acknowledgments

The research reported herein was conducted at SRI International and was supported by the Office of Naval Research under Contract N00014-85-C-0251. The views and conclusions expressed in this document are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Office of Naval Research or the U.S. Government. The author wishes

also to thank AT&T Bell Laboratories for providing the facilities to complete the final draft of this report.

References

[1] Allen, J.F. and J.A. Kooman, "Planning Using a Temporal World Model," *Proc. IJCAI-83*, Karlsruhe, West Germany, pp 741–747 (August 1983).

[2] Allen, J.F., "Towards a General Theory of Action and Time," *Artificial Intelligence*, Vol. 23, pp 123–154 (1984).

[3] Chapman, D., "Planning for Conjunctive Goals," Technical Report 802, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts (November, 1985).

[4] Cheeseman, P., "A Representation of Time for Automatic Planning," *Proc. IEEE Int. Conf. on Robotics*, Atlanta, Georgia (March 1984).

[5] Currie, K. and A. Tate, "O-Plan: Control in the Open Planning Architecture," Report AIAI-TR-12, Artificial Intelligence Applications Institute, University of Edinburgh, Edinburgh, Scotland (1985).

[6] Dean, T., "Temporal Imagery: An Approach to Reasoning about Time for Planning and Problem Solving," Research Report 433, Computer Science Department, Yale University, New Haven, Connecticut (October 1985).

[7] Fikes, R.E. and N.J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, Vol 2, pp 189–208 (1971).

[8] Fikes, R.E., Hart, P. and N.J. Nilsson, "Learning and Executing Generalized Robot Plans," *Artificial Intelligence*, Vol. 3, No. 4, pp 251–288 (1972).

[9] Georgeff, M.P., "A Theory of Action for Multiagent Planning," *Proc. AAAI-84*, Austin, Texas, pp 121–125 (August, 1984).

[10] Georgeff, M.P. and A.L. Lansky, "A System for Reasoning in Dynamic Domains: Fault Diagnosis on the Space Shuttle," Technical Note 375, Artificial Intelligence Center, SRI International, Menlo Park, California (January 1986).

[11] Georgeff, M.P., "Actions, Processes, and Causality," in M.P. Georgeff and A.L. Lansky (eds.), *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop* (Morgan Kaufmann, Los Altos, California, 1987).

[12] Georgeff, M.P., "The Representation of Events in Multiagent Domains," *Proc. AAAI-86*, Philadelphia, Pennsylvania, pp 70–75 (August 1986).

[13] Harel, D., "First-Order Dynamic Logic," *Lecture Notes in Computer Science*, Vol. 68, G. Goos and J. Hartmanis eds. (Springer-Verlag, New York, New York, 1979).

[14] Hayes, P., "The Frame Problem and Related Problems in Artificial Intelligence," in *Artificial and Human Thinking*, A. Elithorn, and D. Jones eds., pp 45–59 (Jossey-Bass, 1973).

[15] Hendrix, G.G., "Modeling Simultaneous Actions and Continuous Processes," *Artificial Intelligence*, Vol. 4, pp 145–180 (1973).

[16] Lansky, A.L., "Behavioral Specification and Planning for Multiagent Domains," Technical Note 360, Artificial Intelligence Center, SRI International, Menlo Park, California (1985).

[17] Lansky, A.L., "A Representation of Parallel Activity Based on Events, Structure, and Causality," in M.P. Georgeff and A.L. Lansky (eds.), *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop* (Morgan Kaufmann, Los Altos, California, 1987).

[18] Lifschitz, V., "On the Semantics of STRIPS," in M.P. Georgeff and A.L. Lansky (eds.), *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop* (Morgan Kaufmann, Los Altos, California, 1987).

[19] Manna, Z., and R. Waldinger, *The Logical Basis for Computer Programming, Volume I: Deductive Reasoning* (Addison-Wesley, Reading, Massachusetts, 1985).

[20] McCarthy, J. and P. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in *Machine Intelligence 4*, B. Meltzer and D. Michie eds., pp 463–502 (Edinburgh University Press, Edinburgh, Scotland, 1969).

[21] McDermott, D., "A Temporal Logic for Reasoning about Processes and Plans," *Cognitive Science*, Vol. 6, pp 101–155 (1982).

[22] McDermott, D., "Generalizing Problem Reduction: A Logical Analysis," *Proc. IJCAI-83*, Karlsruhe, West Germany, pp 302–308 (August 1983).

[23] McDermott, D., "Reasoning about Plans," in *Formal Theories of the Commonsense World*, Hobbs, J.R. and R.C. Moore eds., pp 269–317 (Ablex Publishing, Norwood, New Jersey, 1985).

[24] Pednault, E.P.D., "Preliminary Report on a Theory of Plan Synthesis," Technical Report 358, Artificial Intelligence Center, SRI International, Menlo Park, California (August 1985).

[25] Pednault, E.P.D., *Toward a Mathematical Theory of Plan Synthesis*, Ph.D. thesis, Department of Electrical Engineering, Stanford University, Stanford, California.

[26] Sacerdoti, E.D., "A Stricture for Plans and Behavior," Technical Note 109, Artificial Intelligence Center, SRI International, Menlo Park, California (August 1975).

[27] Shoenfield, J.R., *Mathematical Logic* (Addison-Wesley, Reading, Massachusetts, 1967).

[28] Stefik, M., "Planning With Constraints (MOLGEN: Part 1)," *Artificial Intelligence*, Vol. 16, No. 2, pp 111–140 (May 1981).

[29] Stuart, C, "An Implementation of a Multi-Agent Plan Synchronizer," *Proc. IJCAI-85*, University of California at Los Angeles, Los Angeles, California, pp 1031–1033 (August 1985).

[30] Stuart, C, "A New View of Parallel Activity for Conflict Resolution," in M.P. Georgeff and A.L. Lansky (eds.), *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop* (Morgan Kaufmann, Los Altos, California, 1987).

[31] Suppes, P., *Axiomatic Set Theory* (Dover, New York, New York, 1972).

[32] Sussman, G.J., "A Computational Model of Skill Aquisition," Report AI TR-297, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts (August 1973).

[33] Tate, A., "Goal Structure, Holding Periods and Clouds," in M.P. Georgeff and A.L. Lansky (eds.), *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop* (Morgan Kaufmann, Los Altos, California, 1987).

[34] Van Dalen, D., *Logic and Structure* (Springer-Verlag, Berlin, Germany, 1980).

[35] Vere, S., "Planning in Time: Windows and Durations for Activities and Goals," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol 5., No. 3, pp 246–267 (May 1983).

[36] Vere, S., "Temporal Scope of Assertions and Window Cutoff," *Proc. IJCAI-85*, University of California at Los Angeles, Los Angeles, California, pp 1055–1059 (August 1985).

[37] Vilain, M.B., "A System for Reasoning about Time," *Proc. AAAI-82*, Pittsburgh, Pennsylvania, pp 197–201 (August 1982).

[38] Waldinger, R., "Achieving Several Goals Simultaneously," in, *Machine Intelligence 8*, E. Elcock and D. Michie eds., pp 94–136 (Ellis Horwood, Edinburgh, Scotland, 1977).