ADL and the State-Transition Model of Action

EDWIN P. D. PEDNAULT, *AT&T Bell Laboratories, 101 Crawfords Corner Road, Holmdel, NJ 07733-3030, USA.*
E-mail: epdp@research.att.com

Abstract

This paper provides a complete presentation of the syntax, semantics, and some of the properties of ADL, a formalism for representing and reasoning about the effects of actions. ADL is based on the state-transition model of action, combining much of the expressive power of the situation calculus with the notational and computational benefits of the STRIPS language. In combining these elements, ADL attempts to strike a better balance in the tradeoff between the expressiveness of a logical formalism and the computational complexity of reasoning with that formalism. ADL is defined from a semantic standpoint as a set of constraints on the general state-transition model of action. These constraints are general enough to enable situation-dependent effects to be described while being strong enough to provide a simple solution to the frame problem. The syntax of ADL can vary, allowing different syntactical variants to be developed without altering the underlying semantics. The syntax presented here resembles the STRIPS operator language augmented with conditional add and delete lists. Reasoning is accomplished by transforming ADL schemas into regression operators. It is shown that regression operators provide a sound and complete means for reasoning about the effects of actions represented in ADL. It is also shown that, in general, comparable progression operators cannot be constructed from ADL schemas. This result is surprising, since progression operators are conceptually the inverse of regression operators. Reasoning can also be accomplished by axiomatizing ADL schemas in the situation calculus. It is shown that a restricted form of the situation calculus can be defined that is effectively equivalent to ADL in that ADL schemas can be translated into equivalent sets of axioms in this restricted form, and vice versa. The restricted form is interesting in its own right, since it incorporates a solution to the frame problem that does not explicitly rely on circumscription. This solution is derived from the semantics of ADL.

Keywords: Knowledge representation, reasoning about action, automatic planning, situation calculus, STRIPS, ADL.

1 Introduction

Ever since the work of Newell *et al.* [52, 13] on the GPS problem solver, and the work of McCarthy [44] on the advice-taker program, the state-transition model of action has been the principal framework for representing and reasoning about actions and their effects. In this model, the world is viewed as being in some state of affairs and an action is modelled as a transition of the world from one state to another.

Several formalisms have been developed based on the state-transition model. Two of these are the situation calculus [45] and the STRIPS operator language [14]. The situation calculus is a discipline for axiomatizing the effects of actions in first-order logic and was developed as outgrowth of McCarthy's advice-taker effort. In the situation calculus, states of affairs are represented as individual objects called *situations*. Actions are also represented as individual objects, while state transitions are represented by a function that maps actions and situations to new situations. The effects of actions are axiomatized in terms of this function.

Actions are represented quite differently in the STRIPS language. STRIPS operators define the effects of actions in terms of transformations on sets of formulas. Given a set of formulas that describes the state of affairs that exists before an action is performed, a STRIPS operator for that

action will transform the set into another set of formulas that describes the state of affairs after the action is performed. Functions that define such transformations are called *progression operators* (after Rosenschein [67]). The STRIPS language provides a means of describing progression operators of a particular form.

The situation calculus and the STRIPS language tend to be used for very distinct purposes. Because the situation calculus enables the full expressive power of first-order logic to be employed, it has been used extensively in theoretical work on representing and reasoning about action and change. On the other hand, the situation calculus has seen very little use in implemented systems. This is particularly true in automatic planning, where one reasons about the effects of actions in order to construct plans. Instead, planning systems have typically employed representations based on the STRIPS operator language.

In automatic planning, the preference of STRIPS over the situation calculus is primarily the result of the expressiveness-tractability tradeoff [34, 35]. With expressiveness comes computational intractability—the more expressive a language is, the more computation is required in general to reason about sentences in that language. Reasoning about actions in the situation calculus reduces to theorem proving in first-order logic, which in the general case is only semidecidable (e.g. [70, 8, 74]). As Green [19] demonstrated, the general intractability of theorem proving in first-order logic makes it difficult to efficiently solve planning problems using the situation calculus and general-purpose theorem-proving techniques.[1] The STRIPS operator language was developed in response to Green's results as a means of overcoming some of the computational barriers posed by the situation calculus [14]. The STRIPS language is not as expressive as the situation calculus, which simplifies the reasoning task and makes it possible to develop efficient reasoning algorithms.

The expressiveness of the STRIPS language is constrained by the kinds of transformations on sets of formulas that can be described in the language. With STRIPS operators, transformations on sets of formulas are accomplished by deleting certain formulas from the set to be transformed and then adding certain additional formulas. For a given STRIPS operator, the formulas to be added and deleted are fixed for all sets of formulas to be transformed. Consequently, STRIPS operators cannot adequately model actions whose effects depend on the situations in which they are performed. For example, consider the action of firing the engines of a spacecraft for a given amount of time. The resulting change in trajectory depends not only on the duration of the burn, but also on the position, velocity, mass, and orientation of the spacecraft at the time of firing and on the thrust and fuel consumption of the engines. This action can be axiomatized in the situation calculus (see Section 8); however, it cannot be modelled by means of a STRIPS operator because the formulas that would have to be added and deleted would depend on the set of formulas to be transformed.

Although efficient reasoning is possible when employing the STRIPS language, it is generally recognized that the expressiveness of STRIPS is inadequate for modeling actions in many real-world applications. This inadequacy motivated the development of the ADL language [53, 54, 55, 58]. ADL, which stands for Action Description Language, lies between the STRIPS language and the situation calculus in terms of expressiveness. Its expressive power is sufficient to allow the spacecraft example described above to be represented, yet at the same time it is restrictive enough to allow efficient reasoning algorithms to be developed. In particular, several planning techniques has been developed that take advantage of the properties of ADL. The original

[1] It is possible, however, to efficiently solve planning problem using highly expressive logics by employing special-purpose theorem-proving techniques called *tactics* [7, 5]. In this case, the tradeoff is between tractability and completeness—not all planning problems that can be expressed in the logic can be solved using the special-purpose tactics of the theorem prover.

technique developed by the author [53, 54, 56, 57, 59] has been independently implemented by McDermott [49], Collins and Pryor [10], and Penberthy and Weld [60] who have made additional refinements to improve its efficiency. These planning techniques have also been significantly extended to handle certain forms of continuous processes [61, 62], actions with probabilistic effects [12, 32, 33], conditional branching [64, 12], and knowledge-producing actions [12]. The original planning technique is actually independent of the ADL language and was developed to provide a general approach for synthesizing plans that contain actions with context-dependent effects. ADL complements the technique by allowing the necessary goal expressions for dealing with context-dependent effects to be readily constructed from descriptions of the effects of actions. The implementations mentioned above exploit this property of ADL.

Interestingly, ADL is effectively equivalent to a restricted form of the situation calculus in that any action described in ADL has an equivalent axiomatization in this restricted form, and vice versa. This equivalence is significant because the semantics of ADL provides a solution to the *frame problem* [45, 25, 9] that does not explicitly rely on *circumscription* [46, 22, 23, 31, 36, 37, 38, 39, 71, 2, 3] or other forms of non-monotonic reasoning. In the situation calculus, it is necessary to axiomatize not only the changes that take place when an action is performed, but also the conditions that remain unchanged. Axioms that define changes are called *effect axioms*, while those that specify what remains unchanged are referred to as *frame axioms*. Typically, many more frame axioms are needed than effect axioms, making them cumbersome to write. In addition, one would intuitively expect it to be possible to infer frame axioms from effect axioms. The crux of the frame problem is to accomplish this inference. The inference is necessarily non-monotonic in that axiomatizing additional effects is likely to change the frame axioms that need to be inferred. Techniques such as circumscription attempt to infer frame axioms by minimizing the number of changes that take place when actions are performed. However, different ways of measuring the number of changes yield different frame axioms, not all of which are intuitively satisfying [22, 23, 31, 71, 2, 3, 39]. In addition, there can be several minima and, hence, no uniquely defined set of frame axioms. The ADL language, on the other hand, is sufficiently constrained so that in effect there is a unique and obvious minimum. This allows the frame problem to be readily solved. In actual fact, this is not how the frame problem is approached in ADL; however, the analogy is useful in understanding the connection between the solution incorporated into the semantics of ADL and other approaches to the frame problem.

ADL's solution to the frame problem has a direct counterpart in the restricted form of the situation calculus presented here. The restricted form likewise inherits the computational benefits of ADL. These two properties make the restricted form interesting in its own right. For example, these properties motivated Reiter [65] to adopt the restricted form with slight modifications that incorporate the ideas of Haas [21] and Schubert [69]. The resulting variant has been applied to problems outside the area of automatic planning [66]. Scherl and Levesque [68] have applied the solution to the frame problem embodied in the restricted form to Moore's framework for reasoning about knowledge and action [50, 51], yielding a method for automatically generating frame axioms for knowledge-producing actions. In addition, Kartha [29] has shown how the language $\mathcal{A}$ of Gelfond and Lifschitz [17] can be translated into the restricted form in a way that preserves the soundness and completeness of $\mathcal{A}$.

The purpose of this paper is to provide a complete presentation of the syntax, semantics, and some of the properties of ADL. The paper extends and refines earlier presentations and presents some previously unpublished results.

To motivate the semantics of ADL, a general set-theoretic definition of the state-transition model is presented in Section 2 and the kinds of actions that can be modelled in this framework

are discussed [54, 55]. Section 2 focuses on the general properties of the state-transition model, showing how the concept of a state transition can be used in non-standard ways to model dynamic worlds and continuous processes.

The semantics of ADL are presented in Section 3 as a set of constraints on the state-transition model [53, 54, 55]. These constraints define ADL. They also characterize the kinds of actions that can be represented in ADL versus those that can be modelled in the general state-transition framework.

Because ADL is defined from a semantic standpoint, different syntactical variants of ADL are possible. The syntax employed is merely a vehicle for describing actions while ensuring that the semantical constraints are met. Section 4 presents one such syntax for ADL [54, 58]. This syntax resembles the STRIPS operator language augmented with conditional add and delete lists. Although ADL is semantically quite different from STRIPS, the STRIPS syntax is intuitively appealing and is easily extended to provide a convenient language for specifying ADL schemas.

The problem of reasoning about the effects of actions represented in ADL is the topic of Section 5. In particular, it is shown how *regression operators* [75] can be constructed from ADL schemas. Regression operators are functions from formulas to formulas that conceptually perform the inverse operation of progression operators. Given a formula that describes what is true after an action is performed, the regression of that formula describes what must have been true immediately before the action was performed. Regression operators enable one to transform the problem of determining whether a given condition is true after executing a sequence of actions into the problem of determining whether the appropriate preconditions are true initially. Equations are presented for computing regressions that generalize similar equations presented previously [53, 54, 58]. A rigorous proof is presented showing that the revised equations provide a sound and complete means for reasoning about the effects of actions represented in ADL.

Sections 6 and 7 present results that have not previously appeared in the open literature [54]. Section 6 presents several theorems regarding regression operators and their properties. Section 7 compares ADL to the general class of progression operators, of which STRIPS operators are a special case. It is shown that there are actions representable in ADL whose corresponding progression operators would have to transform finite formulas into infinite sets of formulas in order to provide a sound and complete means of reasoning about the effects of those actions. The corresponding regression operators, on the other hand, always preserve finiteness while providing soundness and completeness. This result is surprising, since progression operators are conceptually the inverse of regression operators. Because of this result, regression operators are preferred over progression operators when using ADL.

Finally, Section 8 compares ADL to the situation calculus, showing how ADL is effectively equivalent to a restricted form of the situation calculus [54, 58]. The syntactic constraints of this restricted form provide another characterization of the kinds of actions that can be represented in ADL. In turn, the semantics of ADL provide a solution to the frame problem for this restricted form that does not explicitly rely on circumscription.

2 The state-transition model of action

As mentioned in the introduction, the state-transition model of action assumes that the world may be in any one of a potentially infinite number of states. The effect of an action is to cause the world to make a transition from one state to another. For example, the action of walking through a doorway from one room to another corresponds to a transition from the state of being in the first room to the state of being in the second. Such actions can be characterized as sets of

current-state/next-state 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'. These pairs summarize the possible state transitions that can occur when the corresponding actions are performed. Stated formally,

DEFINITION 2.1
An *action* over a set of states $\mathcal{W}$ is a binary relation $a \subseteq \mathcal{W} \times \mathcal{W}$ such that $\langle s, t \rangle \in a$ if and only if it is possible for a transition to occur from state s to state t when action a is performed in state s.

DEFINITION 2.2
A *state-transition model* is a pair $\langle \mathcal{W}, \mathcal{A} \rangle$, where $\mathcal{W}$ is a set of possible states of the world and $\mathcal{A}$ is a set of actions over those states.

Note that the above definitions admit both deterministic and non-deterministic actions. If there is at most one next-state t for any current-state s, $\langle s, t \rangle \in a$, then action a is deterministic. In the case of a non-deterministic action, some current-states s have several possible next-states t that the world could transition to when the action is performed. Moreover, which transition occurs can change randomly in repeated trials in which the action is performed several times from state s (after performing other actions as necessary to bring the world back to s). An example would be the action of flipping a coin. This action has two possible outcomes—heads or tails—that cannot be predicted from one coin toss to the next for all intents and purposes.

The state-transition model also encodes the conditions under which actions can be executed. An action a can be performed in a state s if and only if there is some next-state t such that $\langle s, t \rangle \in a$. 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, there would be no corresponding next state. The set of states in which an action a can be executed is given by the domain of the binary relation a:

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

2.1 *Modelling real-world actions*

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? Historically, the state-transition model of action was used to formalize planning problems that involve single agents operating in environments that remains essentially static, except while the agents are performing actions. When an action is performed, the environment undergoes continuous change until the action has been completed. Thereafter, the environment remains in a static configuration until the initiation of the next action. To plan a sequence of actions to achieve a particular configuration, all that one needs to know are the transitions that occur from one static configuration of the environment to another when actions are performed. The continuous changes that take place during execution are not relevant for planning purposes. These observations give rise to the state-transition model. The states effectively catalogue all possible static configurations of the environment, while the current-state/next-state pairs define the transitions that take place when the corresponding actions are performed.

Unfortunately, in many real-world situations the operating environment is never static, but undergoes continuous change. For example, consider the problem of landing a spacecraft on the moon. At any point in time, the vehicle's flight path may be changed by adjusting the attitude of the vehicle and/or its thrust. To plan the flight adjustments, we must deal with the fact that the vehicle undergoes continuous motion until it lands (safely, we hope) on the lunar surface.

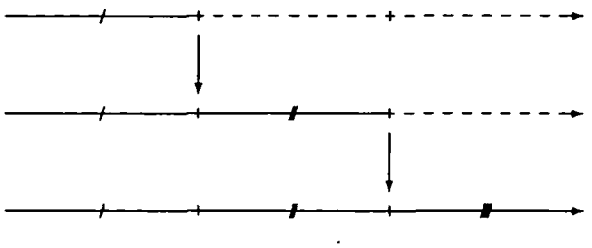

FIG. 1. Interpreting states as chronicles

Although it might not seem possible at first, dynamic-world problems such as this can be formulated using the state-transition model of action. The formulation is accomplished by assigning new interpretations to the mathematical notions of states and state transitions. The key is to use states not for cataloguing static configurations of the environment, but rather for cataloguing entire courses of events mapped onto a time line. To use McDermott's terminology [47, 48], 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.

Although McDermott developed his notion of a chronicle in the context of a temporal logic with the intention of going beyond the capabilities of the state-transition model, his concept can nevertheless be incorporated into the state-transition model by interpreting states as chronicles. If we define states to be chronicles, a state transition then corresponds to the initiation of an action or course of action at a particular point in time. Thus, when an action is initiated and a state transition takes place 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 inaccessible 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 initiating the action that the state transition represents. Initiating a subsequent action would then change these future events. Note that the time of initiation must 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 transitions is illustrated in Fig. 1. States (i.e. chronicles) are shown as time lines, while state transitions are depicted as transitions from one time line to another that has the same past but a different future. 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, so that a new chronicle of events comes into being. 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 [47, 48]. This isomorphism becomes evident from the fact that each of McDermott's forward-branching trees can be expanded into a set of 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.

Thus far, we have considered single agents acting alone in some environment. The real world, however, contains many agents capable of performing tasks simultaneously. Can we model the effects of simultaneous actions using the state-transition model? In attempting to do so, we are faced with a major incongruity: actions in the state-transition model may be performed only sequentially. In some situations, the effect of executing actions simultaneously is the same as doing them sequentially, regardless of the order of execution. This can happen, for example, when the actions affect independent parts of the world and so their effects do not interact. If such is the case, we can reason about the effects of simultaneous actions by imposing a sequential order and then applying the state-transition model.

However, 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 might slide around and perhaps even fall off. On the other hand, if the ends are lifted simultaneously and with equal force, the objects will remain more or less fixed relative to the table.

On the surface, it would appear that synergistic simultaneous actions cannot be dealt with using the state-transition model, since the latter requires that actions be performed sequentially. However, by choosing an appropriate representation, some synergistic simultaneous actions can be made to look like sequential actions, thereby permitting their formulation as state transitions. In the table example described above, this property can be achieved by modelling the forces that are acting upon on the objects. The overall motion of the objects is then determined by the sum of the applied forces and the physical constraints. In this approach, 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/time line and each state transition represents the effect of adding or subtracting a force vector at a particular instant in time. When a transition occurs, not only is the future portion of the chronicle modified, but a record is also 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, the boundary conditions are the forces applied to the table. When actions are performed simultaneously, they add to and modify each other's boundary conditions. The resulting boundary conditions then determine the net effects of the actions. Furthermore, the boundary conditions 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 using the state-transition model of action.

The approach outlined above extends the ideas of Hendrix [26]. A variant of the approach has been implemented in the Zeno planner of Penberthy and Weld [61, 62]. A different approach to modelling simultaneous actions is being pursued by Georgeff [18], Belegrinos and Georgeff [6], Gelfond *et al.* [16], Lin and Shoham [40], and Baral and Gelfond [4]. In these approaches, collections of distinct actions to be performed simultaneously are combined into single composite actions that reflect the simultaneous execution of the ensembles. Each approach generally provides

ways of easily combining the effects of actions that affect independent parts of the world. They then offer different approaches for specifying the combined effects of synergistic actions. In many cases, the effects of different combinations of synergistic actions need to be axiomatized individually. A third approach is being explored by Große [20] that combines aspects of the other two. In this approach, actions give rise to events, and events can in turn cause further events. Synergistic effects are modeled by specifying how the primitive events caused by the actions combine to produce different consequential events.

2.2 *Describing states using first-order languages*

The state-transition model of action clearly allows many actions in the real world to be represented. However, the set-theoretic definition of the model does not lend itself to direct use in practice. In general, the set $\mathcal{W}$ of possible states of the world can be infinite or at least so large that it is not practical to enumerate all the states. In such cases, states must be dealt with indirectly via facts about them expressed in a formal language. For example, the lunar landing problem described earlier involves uncountably many states; however, only a page or two of mathematical formulas is required to define the problem precisely. The STRIPS language and the situation calculus use formulas of first-order languages to describe states. The same is done in ADL.

If first-order formulas are used to describe facts about states, a formula must either be true or false with respect to a state. States must therefore correspond to the semantic structures (i.e. Tarskian models, interpretations, algebraic structures, etc.) that form the basis for the model theory of first-order logic. The only difference between the definition of a state needed here and the usual definition of such structures (e.g. [70, 8, 74]) is that we require states to also assign interpretations to free variables in addition to the other symbols in a first-order language.

In first-order logic, free variables are usually treated as though they are universally quantified. In automatic planning, on the other hand, free variables (called *formal objects* [72]) are used in a way that requires them to have specific values. In particular, free variables are often used to defer the choice of precisely which objects will participate in an action when several options are available. By using variables as place holders, the choice can be deferred until more of the plan is constructed and a better assessment can be made as to the most appropriate objects. If one of these variables appears at multiple points in a plan, it is meant to represent the same object at each of those points.

To use variables in this fashion, states must assign values to variable symbols and actions must be required to preserve these values. It is worth noting that Manna and Waldinger [42, 43] also find it useful to assign values to variable symbols in their presentation of first-order logic, making their semantic structures equivalent to the definition of a state that will now be introduced.

If s is a state, we will write $\mathrm{Dom}(s)$ to mean the domain of that state (i.e. the non-empty set of objects that exist in the world when the world is in that state). If R is a relation symbol, F a function symbol, and x a variable symbol, we will write $s(R)$, $s(F)$, and $s(x)$ to mean the interpretations of R, F, and x, respectively, in state s. Thus, for n-ary relation symbols R, $s(R)$ is a set of n-tuples of the form $\langle d_1, \ldots, d_n \rangle$, where $d_1, \ldots, d_n \in \mathrm{Dom}(s)$. For relation symbols P with zero arguments (i.e. *propositions*), P is true if $s(P) = \{\langle\rangle\}$ and false if $s(P) = \{\} = \emptyset$. For function symbols F of n arguments, $s(F)$ is a set of $(n+1)$-tuples of the form $\langle d_1, \ldots, d_n, d_{n+1} \rangle$, where $d_1, \ldots, d_{n+1} \in \mathrm{Dom}(s)$ and d_{n+1} is the value of the function $s(F)$ with arguments $d_1, \ldots, d_n$. As in the standard model theory of first-order logic, the function $s(F)$ must be defined for all values of its arguments in $\mathrm{Dom}(s)$. Functions can likewise have zero arguments, in which case $s(F)$ is simply an element of $\mathrm{Dom}(s)$. Such symbols

are called *constant symbols* in first-order logic, but here they are treated as non-rigid designators whose values can change from state to state. For example, in a block-stacking problem, we could introduce the symbol *LastBlockMoved* whose interpretation is the block that was moved by the last block-stacking action. In the case of a variable symbol x, the interpretation $s(x)$ of x is likewise an element of $\mathrm{Dom}(s)$.

As usual, objects are referred to by means of terms, which are either variable symbols, function symbols of zero arguments, or function symbols of one or more arguments applied to a list of terms. For convenience, we will extend our notation so that $s(\tau)$ denotes the interpretation of term τ in state s. For variable symbols and function symbols of zero arguments, $s(\tau)$ is defined as above. For function symbols of one or more arguments,

$$s\big(F(\tau_1, \ldots, \tau_n)\big) = y \text{ for } y \text{ such that } \langle s(\tau_1), \ldots, s(\tau_n), y\rangle \in s(F).$$

As in the standard model theory of first-order logic, states assign truth values to formulas by virtue of the fact that they assign interpretations to the symbols that appear in formulas. Let $s \models \varphi$ mean that formula φ is true in state s (or, equivalently, that s satisfies φ). Then the truth value of φ is defined recursively as follows:

1. $s \models \mathrm{TRUE}$
2. $s \not\models \mathrm{FALSE}$
3. $s \models (\tau_1 = \tau_2)$ if and only if $s(\tau_1) = s(\tau_2)$
4. $s \models R(\tau_1, \ldots, \tau_n)$ if and only if $\langle s(\tau_1), \ldots, s(\tau_n)\rangle \in s(R)$
5. $s \models \neg\varphi$ if and only if $s \not\models \varphi$
6. $s \models \varphi \wedge \psi$ if and only if $s \models \varphi$ and $s \models \psi$
7. $s \models \varphi \vee \psi$ if and only if $s \models \varphi$ or $s \models \psi$
8. $s \models \varphi \rightarrow \psi$ if and only if $s \not\models \varphi$ or $s \models \psi$
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 $s[d/x] \models \varphi$ for all $d \in \mathrm{Dom}(s)$
11. $s \models \exists x\, \varphi$ if and only if $s[d/x] \models \varphi$ for some $d \in \mathrm{Dom}(s)$

where $s[d/x]$ is the state obtained from s by changing the value of the variable x to be equal to d. Thus, $\mathrm{Dom}(s[d/x]) = \mathrm{Dom}(s)$, and for any symbol σ,

$$s[d/x](\sigma) = \begin{cases} d & \text{if } \sigma \text{ is } x \\ s(\sigma) & \text{otherwise.} \end{cases}$$

To complete the specialization of the state-transition model of action to first-order languages, two other issues also need to be addressed. From a model-theoretic point of view, if s is a semantic structure, then $s[d/x]$ is also a semantic structure. Therefore, within the state-transition model we must require $s[d/x]$ to be a state of the world if s is a state of the world. In addition, free variables are used in automatic planning in a way that requires a variable to refer to the same object in each state of the world that arises during the execution of a sequence of actions. For this reason, we will require that actions preserve the interpretations of variable symbols. Thus, if a is an action and $\langle s, t\rangle \in a$, then it must be the case that $t(x) = s(x)$ for every variable symbol x. These considerations lead to the following definition.

DEFINITION 2.3
A state-transition model for a first-order language L is a state-transition model $\langle \mathcal{W}, \mathcal{A}\rangle$ that satisfies the following conditions:

1. Every state $s \in \mathcal{W}$ is a semantic structure that assigns interpretations to the relation, function, and variable symbols of L.

2. For every state $s \in \mathcal{W}$, every variable symbol x, and every element $d \in \mathrm{Dom}(s)$, it is the case that $s[d/x] \in \mathcal{W}$.

3. Every action $a \in \mathcal{A}$ has the property that $t(x) = s(x)$ for every variable symbol x and every pair of states $\langle s, t \rangle \in a$.

Note that state-transition models for first-order languages are similar in definition to the semantic structures of first-order dynamic logic [63, 24], a logic developed for the purpose of program verification. Actions in the state-transition model correspond to programs in first-order dynamic logic, and state transitions correspond to program executions that terminate. This correspondence has been noted by Rosenschein [67] and Kautz [30], who developed dynamic logics for planning purposes. Definition 2.3, however, is not tied to dynamic logic, but is intended to provide a general definition of the state-transition model suitable for any framework employing first-order languages.

2.3 *Deduction and logical implication*

It should be noted that because states assign values to variable symbols, the standard proof theory of first-order logic cannot be used directly without modification. The reason is that free variables are treated as though they are universally quantified under the standard semantics of first-order logic, but not under the semantics presented here. For example, if R is a unary relation symbol and x is a variable symbol, then $R(x)$ logically implies $\forall x\, R(x)$ under the standard semantics; however, under the semantics presented here, $R(x)$ only imply that R is true of x. This allows $R(y)$ to be false for some $y \neq x$. Consequently, $R(x)$ does not logically imply $\forall x\, R(x)$ under the semantics employed here.

To prove logical implication for the semantics presented here, the proof theory of Manna and Waldinger [42, 43] can be used instead of the standard proof theory, since their semantics are equivalent. However, it is also possible to introduce a preprocessing step so that the standard proof theory of first-order logic can be applied. This can be done as follows. Suppose that we wish to determine whether the formula φ is a theorem of the set of formulas Γ, where φ and Γ may both contain free variables. For every free variable x_i that appears in φ or Γ, introduce a new zero-place function symbol (i.e. 'constant' symbol) c_i into the language. Let φ' and Γ' be the formula and the set of formulas, respectively, that are obtained by replacing every free occurrence of x_i with c_i in φ and Γ for every free variable x_i. Then φ is a logical consequence of Γ according to the semantics presented here if and only if φ' is a logical consequence of Γ' under the standard semantics of first-order logic. Moreover, the latter is true if and only if φ' is provable from Γ' using the standard proof theory of first-order logic. The reason that these relationships hold is that free variables and constant symbols are semantically indistinguishable in the definition of a state presented above. By replacing free variables with new constant symbols, we can simulate the semantics presented here using the standard semantics of first-order logic. This enables us to use the standard proof theory at the cost of first performing a simple transformation on the formulas being compared.

3 The semantics of ADL

The ADL formalism is semantically defined by introducing four additional constraints to Definition 2.3. The purpose of the constraints is to strike a particular balance between the expressiveness of the resulting formalism and the computational effort required to reason about the effects of actions.

The first constraint is that actions may not change the set of objects that exist in the world; that is, for every action a and every current-state/next-state pair $\langle s, t\rangle \in a$, it must be the case that $\mathrm{Dom}(t) = \mathrm{Dom}(s)$. This requirement is of concern only when we wish to model actions that create or destroy objects. Because actions are prevented from adding or deleting objects from the domains of states, the creation and destruction of objects must be modelled 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 $U(x)$. This method, of course, requires that the domain of each state include all objects that could possibly exist. This approach is not unreasonable; for example, this is precisely that way in which memory allocation is performed in computers. New memory is not created 'out of thin air' when an allocation request is made; instead, an area of unused memory is identified and allocated. Likewise, when memory is deallocated, it does not disappear from existence; instead, it is added back to the pool of unused memory. Therefore, this first constraint does not really affect the kinds of actions that can be modeled, it only affects how they are modeled.

The second constraint is that actions in ADL must be deterministic. If $\langle s, t_1\rangle$ and $\langle s, t_2\rangle$ are current-state/next-state pairs of action a, then it must be the case that $t_1 = t_2$. Determinism allows the effects of actions to be more easily described. In particular, the state transitions can be expressed in terms of a collection of functions that specify how the interpretation of each symbol changes when the action is performed. For every relation symbol R we can construct a function f_R^a such that $t(R) = f_R^a(s)$ for every pair of states $\langle s, t\rangle \in a$. Likewise, for every function symbol F we can construct a function f_F^a such that $t(F) = f_F^a(s)$. Together, these functions completely define the effects of action a.

The third constraint incorporated into ADL is that the transformation functions introduced above must be representable as first-order formulas. For every n-ary relation symbol R, there must exist a formula $\varphi_R^a(x_1, \ldots, x_n)$ with free variables $x_1, \ldots, x_n$ such that $f_R^a(s)$ is given by

$$\begin{aligned} t(R) &= f_R^a(s) \\ &= \{\langle d_1, \ldots, d_n\rangle \in \mathrm{Dom}(s)^n \mid s[d_1/x_1, \ldots, d_n/x_n] \models \varphi_R^a(x_1, \ldots, x_n)\}. \end{aligned} \tag{3.1}$$

Thus, $R(x_1, \ldots, x_n)$ will be true after performing action a if and only if $\varphi_R^a(x_1, \ldots, x_n)$ was true just prior to execution. Likewise, for every n-ary function symbol F, there must exist a formula $\varphi_F^a(x_1, \ldots, x_n, y)$ with free variables $x_1, \ldots, x_n, y$ such that $f_F^a(s)$ is given by

$$\begin{aligned} t(F) &= f_F^a(s) \\ &= \{\langle d_1, \ldots, d_n, e\rangle \in \mathrm{Dom}(s)^{n+1} \mid s[d_1/x_1, \ldots, d_n/x_n, e/y] \\ &\qquad \models \varphi_F^a(x_1, \ldots, x_n, y)\}. \end{aligned} \tag{3.2}$$

Consequently, $F(x_1, \ldots, x_n) = y$ will be true after performing action a if and only if $\varphi_F^a(x_1, \ldots, x_n, y)$ was true beforehand. Note that this representability requirement relies on the first constraint that $\mathrm{Dom}(t) = \mathrm{Dom}(s)$.

To illustrate how mappings between interpretations can be represented as formulas, consider the action PutOnBC for placing block B on top of block C. After this action is applied, block

B becomes situated on top of block C and every block except B remains where it was. If the binary relation $\mathrm{On}(x, y)$ is used to represent the fact that block x is on block y, then $\mathrm{On}(x, y)$ will be true immediately after executing PutOnBC if and only if

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

was true just prior to execution. This formula is therefore a suitable value for $\varphi_{\mathrm{On}}^{\mathrm{PutOnBC}}(x, y)$.

The fourth and final constraint incorporated into ADL is that set of states in which an action is executable must also be representable as a formula. For every action a that can be represented in ADL, there must exist a formula π^a with the property that $s \models \pi^a$ if and only if there is some state t for which $\langle s, t\rangle \in a$ (i.e. action a is executable in state s). These formulas are referred to as *executability preconditions*. For example, the executability precondition for placing block B on top of block C might be that there is nothing on top of these two blocks; i.e.

$$\forall z\, \neg\mathrm{On}(z, B) \wedge \forall z\, \neg\mathrm{On}(z, C).$$

This representability constraint for executability conditions is assumed by virtually all action representations that have been developed for automatic planning, including the STRIPS language and the situation calculus. Thus, while it is a bona fide constraint on the general state-transition framework, it does not restrict the kinds of actions that can be described in any practical way relative to other languages in use.

The four constraints presented above provide a complete characterization of the kinds of actions that can be represented in ADL. Given these constraints, an action is completely described by a set of formulas that define the functions for transforming the interpretations of the function and relation symbols from one state to the next, and by a formula that defines the states in which the action can be executed.

3.1 Addressing the frame problem

One of the main contributions of the STRIPS operator language is its solution to the frame problem. The objective of the frame problem is to find a representation for actions that enables one to specify the changes that take place when an action is performed without having to specify everything in the world that remains unaltered. The solution employed in the STRIPS language is to represent the effects of actions as transformations on sets of formulas and to then provide a simple means of specifying such transformations in terms of differences between sets. Because the transformations are completely defined by set differences, there is no need to specify the elements of the sets that are in common.

The semantics of ADL is also based on transformations on sets, but in this case the sets are the semantic interpretations of function and relation symbols rather than sets of formulas. Nevertheless, the same approach to the frame problem can be employed without affecting the kinds of actions that can be represented.

In particular, each state-transformation formula $\varphi_R^a(x_1, \ldots, x_n)$ for a relation symbol R can be expressed in terms of two other formulas, $\alpha_R^a(x_1, \ldots, x_n)$ and $\delta_R^a(x_1, \ldots, x_n)$, that, respectively, describe the additions to and the deletions from the interpretation of R. Thus, if action a is performed in state s, the interpretation of R in the succedent state t is given by

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

where

$$A_R \;=\; \big\{\langle d_1, \ldots, d_n\rangle \in \mathrm{Dom}(s)^n \mid s[d_1/x_1, \ldots, d_n/x_n] \models \alpha_R^a(x_1, \ldots, x_n)\big\}$$

$$D_R \quad = \quad \{\langle d_1, \ldots, d_n\rangle \in \mathrm{Dom}(s)^n \mid s[d_1/x_1, \ldots, d_n/x_n] \models \delta_R^a(x_1, \ldots, x_n)\}.$$

A_R is the set of tuples to be added to $s(R)$ as defined by φ_R^a, and D_R is the set of tuples to be deleted from $s(R)$ as defined by δ_R^a. The formula α_R^a is referred to as the *add conditions* for relation R and action a, while δ_R^a is referred to as the *delete conditions*. These formulas are assumed to completely describe the changes made to relation R. Any change not so defined is presumed not to take place.

For the sake of symmetry, we will require that A_R and D_R be non-intersecting sets (i.e. $A_R \cap D_R = \emptyset$) so that additions and deletions may be done in any order. Then

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

This requirement in turn implies that the add and delete conditions cannot be true simultaneously when the action is performed. Consequently, it must be the case that

$$\pi^a \models \neg\exists x_1, \ldots, x_n\big(\alpha_R^a(x_1, \ldots, x_n) \wedge \delta_R^a(x_1, \ldots, x_n)\big).$$

Given formulas α_R^a and δ_R^a as defined above, the state-transformation formula φ_R^a is given by

$$\alpha_R^a(x_1, \ldots, x_n) \vee \big(R(x_1, \ldots, x_n) \wedge \neg\delta_R^a(x_1, \ldots, x_n)\big). \tag{3.3}$$

Paraphrased in English, this formula states that $R(x_1, \ldots, x_n)$ will be true after performing action a if and only if action a makes it true, or it was true beforehand and a does not make it false. Since α_R^a and δ_R^a define non-intersecting sets, φ_R^a could alternatively be given by

$$\big(R(x_1, \ldots, x_n) \vee \alpha_R^a(x_1, \ldots, x_n)\big) \wedge \neg\delta_R^a(x_1, \ldots, x_n).$$

Note that it is also possible to solve the reverse problem of finding appropriate add and delete conditions given a particular state-transformation formula φ_R^a. For example, we can let $\alpha_R^a(x_1, \ldots, x_n)$ be the formula $\varphi_R^a(x_1, \ldots, x_n)$ and let $\delta_R^a(x_1, \ldots, x_n)$ be $\neg\varphi_R^a(x_1, \ldots, x_n)$. Thus, we may conclude that together α_R^a and δ_R^a provide the same level of expressiveness as the single transformation formula φ_R^a.

As an example, consider again the action PutOnBC of placing block B on top of block C. A suitable add condition $\alpha_{\mathrm{On}}^{\mathrm{PutOnBC}}(x, y)$ for the On relation would be $(x = B \wedge y = C)$, and a suitable delete condition $\delta_{\mathrm{On}}^{\mathrm{PutOnBC}}(x, y)$ would be $(x = B \wedge y \neq C)$. Note that $\delta_{\mathrm{On}}^{\mathrm{PutOnBC}}(x, y)$ cannot simply be $(x = B)$, since we require that the sets defined by $\alpha_{\mathrm{On}}^{\mathrm{PutOnBC}}(x, y)$ and $\delta_{\mathrm{On}}^{\mathrm{PutOnBC}}(x, y)$ be non-intersecting.

The transformation formulas for function symbols can be restructured in a similar manner. However, in the case of function symbols, we can take advantage of the fact that a function must be defined everywhere as required by the semantics of first-order logic. This implies that $F(x_1, \ldots, x_n) = y$ will be true after an action has been performed if and only if the action changed the value of $F(x_1, \ldots, x_n)$ to y, or the action preserved its value and $F(x_1, \ldots, x_n) = y$ was true previously. These changes can therefore be described by a single formula $\mu_F^a(x_1, \ldots, x_n, y)$ that is true if and only if the value of $F(x_1, \ldots, x_n)$ is to be changed to y when action a is performed. The formula μ_F^a is referred to as the *update conditions* for function F and action a. If action a is performed in state s, the interpretation of F in the succedent state t is given by

$$t(F) = (s(F) - D_F) \cup A_F,$$

where

$$A_F = \{\langle d_1, \ldots, d_n, e\rangle \in \mathrm{Dom}(s)^{n+1} \mid s[d_1/x_1, \ldots, d_n/x_n, e/y] \models \mu_F^a(x_1, \ldots, x_n, y)\}$$

$$D_F = \{\langle d_1, \ldots, d_n, e\rangle \in \mathrm{Dom}(s)^{n+1} \mid s[d_1/x_1, \ldots, d_n/x_n, e/y] \models \exists y\,[\mu_F^a(x_1, \ldots, x_n, y)]\}.$$

Since functions have unique values, μ_F^a must have the property that, for any assignment of values to $x_1, \ldots, x_n$, either there is a unique value of y for which $\mu_F^a(x_1, \ldots, x_n, y)$ is true when action a is executed, or there are no such values for y. This requires that

$$\pi^a \models \neg\exists x_1, \ldots, x_n, y, y'\big(\mu_F^a(x_1, \ldots, x_n, y) \wedge \mu_F^a(x_1, \ldots, x_n, y') \wedge y \neq y'\big).$$

Given such an update condition μ_F^a, the state-transformation formula φ_F^a for the function F is given by

$$\mu_F^a(x_1, \ldots, x_n, y) \vee \big(F(x_1, \ldots, x_n) = y \wedge \neg\exists y\, \mu_F^a(x_1, \ldots, x_n, y)\big). \tag{3.4}$$

As with add and delete conditions, an appropriate update condition μ_F^a can be constructed given any state-transformation formula φ_F^a. For example, $\mu_F^a(x_1, \ldots, x_n, y)$ can be $\varphi_F^a(x_1, \ldots, x_n, y)$ itself. Thus, μ_F^a provides an equivalent means of describing the effect of action a on the function F. Note that μ_F^a must completely describe the changes made to the function F, since any change not so defined is presumed not to take place.

3.2 *ADL schemas*

The semantic constraints and decompositions presented above enable the following definitions to be made.

DEFINITION 3.1
An *ADL schema* for an action a consists of

1. A formula π^a that defines the preconditions of execution of action a.
2. A pair of formulas $\alpha_R^a(x_1, \ldots, x_n)$ and $\delta_R^a(x_1, \ldots, x_n)$ for each n-ary relation symbol R that define the add and delete conditions for R.
3. A formula $\mu_F^a(x_1, \ldots, x_n, y)$ for each n-ary function symbol F that defines the update conditions for F.

DEFINITION 3.2
An ADL schema for an action a is said to be *consistent* if and only if for every n-ary relation symbol R,

$$\pi^a \models \neg\exists x_1, \ldots, x_n\big(\alpha_R^a(x_1, \ldots, x_n) \wedge \delta_R^a(x_1, \ldots, x_n)\big),$$

and for every n-ary function symbol F,

$$\pi^a \models \neg\exists x_1, \ldots, x_n, y, y'\big(\mu_F^a(x_1, \ldots, x_n, y) \wedge \mu_F^a(x_1, \ldots, x_n, y') \wedge y \neq y'\big).$$

DEFINITION 3.3
Given a set of states of the world $\mathcal{W}$, the action a defined by an ADL schema is the set of all current-state/next-state pairs $\langle s, t\rangle \in \mathcal{W} \times \mathcal{W}$ such that

1. $s \models \pi^a$.
2. $\mathrm{Dom}(t) = \mathrm{Dom}(s)$.
3. For every variable symbol x, $t(x) = s(x)$.
4. For every n-ary relation symbol R,

$$t(R) = \{\langle d_1, \ldots, d_n\rangle \in \mathrm{Dom}(s)^n \mid s[d_1/x_1, \ldots, d_n/x_n] \models \alpha_R^a(x_1, \ldots, x_n) \vee (R(x_1, \ldots, x_n) \wedge \neg\delta_R^a(x_1, \ldots, x_n))\}.$$

5. For every n-ary function symbol F,

$$t(F) = \{\langle d_1, \ldots, d_n, e\rangle \in \mathrm{Dom}(s)^{n+1} \mid s[d_1/_1, \ldots, d_n/x_n, e/y] \models \mu_F^a(x_1, \ldots, x_n, y) \vee (F(x_1, \ldots, x_n) = y \wedge \neg\exists y[\mu_F^a(x_1, \ldots, x_n, y)])\}.$$

The above definitions completely characterize ADL. The definitions do not rely on any particular syntax for providing the add, delete, update, and executability conditions (α, δ, μ, and π) that define an action. The reason is to allow different syntactic variants of ADL to be constructed without having to redefine the underlying semantics for each new variant. One need only define how the add, delete, update, and executability conditions are constructed for a given syntax.

3.3 Parametric schemas

The definition of an ADL schema can be further refined to provide a mechanism for defining parametric families of actions. For example, instead of constructing a separate definition for each action of moving a block from one location to another, we could define a parametric family of actions $\mathrm{Put}(b, l)$ for moving any block b to any location l. Terms would then be used as arguments to select particular members of a parametric family. Thus, $\mathrm{Put}(B, C)$ would be the action of placing block B on top of block C, while $\mathrm{Put}(\mathit{LastBlockMoved}, \mathit{Table})$ would be the action of placing whatever block was most recently moved onto the table.

A parametric family is defined by a single ADL schema in which the parameters of the schema appear as free variables in the add, delete, update, and executability conditions. Thus, a parametric family $A(p_1, \ldots, p_k)$ with parameters $p_1, \ldots, p_k$ would be specified by a set of formulas of the forms

$$\alpha_R^{A(p_1,\ldots,p_k)}(x_1, \ldots, x_n, p_1, \ldots, p_k)$$
$$\delta_R^{A(p_1,\ldots,p_k)}(x_1, \ldots, x_n, p_1, \ldots, p_k)$$
$$\mu_F^{A(p_1,\ldots,p_k)}(x_1, \ldots, x_n, y, p_1, \ldots, p_k)$$
$$\pi^{A(p_1,\ldots,p_k)}(p_1, \ldots, p_k).$$

The mapping from specific arguments to specific members of a parametric family is semantically defined by evaluating the arguments in the state in which the action is to be performed and then assigning the values of the arguments to the corresponding parameters.

DEFINITION 3.4
Given a set of states of the world $\mathcal{W}$, a collection of terms $\tau_1, \ldots, \tau_k$, and a parameterized ADL schema $A(p_1, \ldots, p_k)$, the action $A(\tau_1, \ldots, \tau_k)$ is the set of current-state/next-state pairs $\langle s, t \rangle \in \mathcal{W} \times \mathcal{W}$ such that

1. $s[s(\tau_1)/p_1, \ldots, s(\tau_k)/p_k] \models \pi^{A(p_1,\ldots,p_k)}(p_1, \ldots, p_k)$.
2. $\mathrm{Dom}(t) = \mathrm{Dom}(s)$.
3. For every variable symbol x, $t(x) = s(x)$.
4. For every n-ary relation symbol R,

$$t(R) = \{\langle d_1, \ldots, d_n \rangle \in \mathrm{Dom}(s)^n \mid s[d_1/x_1, \ldots, d_n/x_n, s(\tau_1)/p_1, \ldots, s(\tau_k)/p_k] \models \alpha_R^{A(p_1,\ldots,p_k)}(x_1, \ldots, x_n, p_1, \ldots, p_k) \vee (R(x_1, \ldots, x_n) \wedge \neg\delta_R^{A(p_1,\ldots,p_k)}(x_1, \ldots, x_n, p_1, \ldots, p_k))\}.$$

5. For every n-ary function symbol F,

$$t(F) = \{\langle d_1, \ldots, d_n, e \rangle \in \mathrm{Dom}(s)^{n+1} \mid s[d_1/x_1, \ldots, d_n/x_n, e/y, s(\tau_1)/p_1, \ldots, s(\tau_k)/p_k] \models \mu_F^{A(p_1,\ldots,p_k)}(x_1, \ldots, x_n, y, p_1, \ldots, p_k) \vee (F(x_1, \ldots, x_n) = y \wedge \neg\exists y[\mu_F^{A(p_1,\ldots,p_k)}(x_1, \ldots, x_n, y, p_1, \ldots, p_k)])\}.$$

Note that the value of each term τ_i is determined in state s and then assigned to the corresponding parameters p_i before evaluating the add, delete, update, and executability conditions (α, δ, μ, and π).

Definition 3.4 defines the meaning of a parameterized schema from a semantic standpoint. This definition has a syntactic counterpart which involves replacing all free occurrences of the variables used as parameters in the add, delete, update, and executability conditions with the terms used as arguments. This produces the corresponding formulas for the specific action selected. The replacement is accomplished by means of *safe substitution* (e.g. [42, 43]) wherein quantified variables are renamed to avoid potential conflicts between bound and unbound instances of variables. For example, to replace the variable p with the term τ in the formula $\varphi(p)$, a simple form of safe substitution would involve replacing all quantified variables in $\varphi(p)$ that also appear in τ with new variable symbols and only then replacing all free occurrences of p with τ. More sophisticated forms of safe substitution avoid unnecessary renaming of quantified variables.

To express the syntactic counterpart to Definition 3.4, let $[\tau/p]\varphi$ be the safe substitution of τ for p in formula φ. Then the add, delete, update, and executability conditions for the action $A(\tau_1, \ldots, \tau_k)$ from the parametric family $A(p_1, \ldots, p_k)$ are given by

$$\alpha_R^{A(\tau_1,\ldots,\tau_k)}(x_1, \ldots, x_n) \equiv [\tau_1/p_1, \ldots, \tau_k/p_k]\alpha_R^{A(p_1,\ldots,p_k)}(x_1, \ldots, x_n, p_1, \ldots, p_k) \tag{3.5}$$

$$\delta_R^{A(\tau_1,\ldots,\tau_k)}(x_1, \ldots, x_n) \tag{3.6}$$

$$\equiv [\tau_1/p_1, \ldots, \tau_k/p_k]\delta_R^{A(p_1,\ldots,p_k)}(x_1, \ldots, x_n, p_1, \ldots, p_k)$$

$$\mu_F^{A(\tau_1,\ldots,\tau_k)}(x_1, \ldots, x_n, y) \tag{3.7}$$

$$\equiv [\tau_1/p_1, \ldots, \tau_k/p_k]\mu_F^{A(p_1,\ldots,p_k)}(x_1, \ldots, x_n, y, p_1, \ldots, p_k)$$

$$\pi^{A(\tau_1,\ldots,\tau_k)} \tag{3.8}$$

$$\equiv [\tau_1/p_1, \ldots, \tau_k/p_k]\pi^{A(p_1,\ldots,p_k)}(p_1, \ldots, p_k),$$

where '$\equiv$' denotes syntactic equality between formulas. Thus, inserting (3.5)–(3.8) into Definition 3.3 produces the same current-state/nest-state pairs for action $A(\tau_1, \ldots, \tau_k)$ as those defined in Definition 3.4. Equations (3.5)–(3.8) therefore provide an equivalent characterization of $A(\tau_1, \ldots, \tau_k)$ as demonstrated by the following theorem.

THEOREM 3.5
Given a set of states of the world $\mathcal{W}$, a collection of terms $\tau_1, \ldots, \tau_k$, a parameterized ADL schema $A(p_1, \ldots, p_k)$, and the formulas $\alpha_R^{A(\tau_1,\ldots,\tau_k)}(x_1, \ldots, x_n)$, $\delta_R^{A(\tau_1,\ldots,\tau_k)}(x_1, \ldots, x_n)$, $\mu_F^{A(\tau_1,\ldots,\tau_k)}(x_1, \ldots, x_n, y)$, and $\pi^{A(\tau_1,\ldots,\tau_k)}$ given by (3.5)–(3.8), it is the case that $\langle s, t\rangle \in A(\tau_1, \ldots, \tau_k)$ if and only if $\langle s, t\rangle \in \mathcal{W} \times \mathcal{W}$ and

1. $s \models \pi^{A(\tau_1,\ldots,\tau_k)}$.
2. $\mathrm{Dom}(t) = \mathrm{Dom}(s)$.
3. For every variable symbol x, $t(x) = s(x)$.
4. For every n-ary relation symbol R,

$$t(R) = \{\langle d_1, \ldots, d_n\rangle \in \mathrm{Dom}(s)^n \mid s[d_1/x_1, \ldots, d_n/x_n] \models \alpha_R^{A(\tau_1,\ldots,\tau_k)}(x_1, \ldots, x_n) \vee (R(x_1, \ldots, x_n) \wedge \neg\delta_R^{A(\tau_1,\ldots,\tau_k)}(x_1, \ldots, x_n))\}.$$

5. For every n-ary function symbol F,

$$t(F) = \{\langle d_1, \ldots, d_n, e\rangle \in \mathrm{Dom}(s)^{n+1} \mid s[d_1/x_1, \ldots, d_n/x_n, e/y] \models \mu_F^{A(\tau_1,\ldots,\tau_k)}(x_1, \ldots, x_n, y) \vee (F(x_1, \ldots, x_n) = y \wedge \neg\exists y[\mu_F^{A(\tau_1,\ldots,\tau_k)}(x_1, \ldots, x_n, y)])\}.$$

PROOF. The theorem follows from Definition 3.4 and the properties of safe substitution; in particular, $s[s(\tau)/p] \models \varphi$ if and only if $s \models [\tau/p]\varphi$ for any term τ and any variable p [42, 43]. By direct application of this property, each clause in Definition 3.4 is equivalent to the corresponding clause in Theorem 3.5. ∎

Throughout the paper, it is assumed that safe substitution is being used whenever a free variable in a formula is replaced by another term. When a formula is defined whose free variables are explicitly identified in the form of a parameter list, as in $\varphi(x_1, \ldots, x_n)$, then the notation $\varphi(\tau_1, \ldots, \tau_n)$ will be used as shorthand for $[\tau_1/x_1, \ldots, \tau_n/x_n]\varphi(x_1, \ldots, x_n)$.

$\mathrm{Put}(b, l)$;;; *Place block b on top of object l*
Precond: $b \neq l$, $b \neq \mathit{Table}$, $\forall z \neg\mathrm{On}(z, b)$, $\big(l = \mathit{Table} \vee \forall z \neg\mathrm{On}(z, l)\big)$
Add: $\mathrm{On}(b, l)$
$\mathrm{Above}(b, l)$
$\mathrm{Above}(b, z)$ for all z such that $\mathrm{Above}(l, z)$
Delete: $\mathrm{On}(b, z)$ for all z such that $z \neq l$
$\mathrm{Above}(b, z)$ for all z such that $z \neq l \wedge \neg\mathrm{Above}(l, z)$
Update: $\mathit{LastBlockMoved} \leftarrow b$

$\mathrm{Pour}(p, q)$;;; *Pour the contents of vessel p into vessel q*
Update: $\mathrm{Volume}(q) \leftarrow \min\big(\mathrm{Capacity}(q), \mathrm{Volume}(p) + \mathrm{Volume}(q)\big)$
$\mathrm{Volume}(p) \leftarrow \max\big(0, \mathrm{Volume}(p) - \mathrm{Capacity}(q) + \mathrm{Volume}(q)\big)$

$\mathrm{Adjust}(a', \theta', t')$;;; *Set acceleration to a' and attitude to θ' at time t'*
Precond: $t_0 \leq t'$, $\min \leq a' \leq \max$
Update: $t_0 \leftarrow t'$
$a(t) \leftarrow a'$ for all t such that $t \geq t'$
$\theta(t) \leftarrow \theta'$ for all t such that $t \geq t'$
$v_x(t) \leftarrow (a' \sin\theta')(t - t') + v_x(t')$ for all t such that $t \geq t'$
$v_y(t) \leftarrow (g - a' \cos\theta')(t - t') + v_y(t')$ for all t such that $t \geq t'$
$x(t) \leftarrow \frac{1}{2}(a' \sin\theta')(t - t')^2 + v_x(t')(t - t') + x(t')$ for all t such that $t \geq t'$
$y(t) \leftarrow \frac{1}{2}(g - a' \cos\theta')(t - t')^2 + v_y(t')(t - t') + x(t')$ for all t such that $t \geq t'$

FIG. 2. Examples of actions formulated in ADL

4 The syntax of ADL

The syntax of ADL presented below is designed to provide a convenient means of specifying the add, delete, update and executability conditions (α, δ, μ, and π formulas) that define an ADL schema. The actions defined in Fig. 2 illustrate this syntax. The first action $\mathrm{Put}(b, l)$ models the blocks-world action of stacking one block atop another or placing it on the table. The second action $\mathrm{Pour}(p, q)$ is inspired by a mathematical puzzle and models the action of pouring a liquid from one container into another until either the first is emptied or the second is filled to capacity. The third action $\mathrm{Adjust}(a', q', t')$ models the action of adjusting the thrust and attitude of a lunar lander and serves to demonstrate how time and continuous processes can be modelled in the manner discussed in Section 2.

As the examples illustrate, an ADL schema consists of an action name, an optional parameter list, and four optional groups of clauses labelled Precond, Add, Delete, and Update. The Precond group consists of a list of formulas that define the preconditions for the execution of the action. Every formula in the list must be true when the action is performed; hence, the overall precondition π is the conjunction of these formulas. If the list is absent, π is taken to be the formula TRUE, meaning that the action may be performed in every state. For instance, the

Clause	$\hat{\alpha}_R(x_1,\ldots,x_n)/\hat{\delta}_R(x_1,\ldots,x_n)$
$R(\tau_1,\ldots,\tau_n)$	$(x_1=\tau_1\wedge\cdots\wedge x_n=\tau_n)$
$R(\tau_1,\ldots,\tau_n)$ if ψ	$(x_1=\tau_1\wedge\cdots\wedge x_n=\tau_n\wedge\psi)$
$R(\tau_1,\ldots,\tau_n)$ for all $z_1,\ldots,z_k$	$\exists z_1,\ldots,z_k\,(x_1=\tau_1\wedge\cdots\wedge x_n=\tau_n)$
$R(\tau_1,\ldots,\tau_n)$ for all $z_1,\ldots,z_k$ such that ψ	$\exists z_1,\ldots,z_k\,(x_1=\tau_1\wedge\cdots\wedge x_n=\tau_n\wedge\psi)$

TABLE 1. Add/Delete clauses and their meanings

precondition of execution for the action $\mathrm{Pour}(p,q)$ is the formula TRUE, whereas for $\mathrm{Put}(b,l)$ it is

$$(b\neq l)\wedge(b\neq \mathit{Table})\wedge\forall z\,\neg\mathrm{On}(z,b)\wedge\big(l=\mathit{Table}\vee\forall z\,\neg\mathrm{On}(z,l)\big).$$

Add and delete conditions are specified by the Add and Delete groups, respectively. Each group consists of a set of clauses of the forms shown in the left-hand column of Table 1. In this table, R is a relation symbol, $\tau_1,\ldots,\tau_n$ are terms, ψ is a formula, $z_1,\ldots,z_k$ are variable symbols that appear in terms $\tau_1,\ldots,\tau_n$ but do not appear in the parameter list of the action schema, and $x_1,\ldots,x_n$ are variable symbols that are distinct from the variables $z_1,\ldots,z_n$ and that do not appear in $\tau_1,\ldots,\tau_n,\psi$, or the parameter list of the action schema. Each clause in an Add group corresponds to an individual add condition $\hat{\alpha}_R$ for a relation symbol R. The corresponding add conditions are defined in the right-hand column of Table 1. The formula α_R that defines the overall add condition for R is the disjunction of the individual $\hat{\alpha}_R$'s. If no add conditions are specified for R, α_R is taken to be the formula FALSE (i.e. nothing is added to relation R). For example, the overall add condition $\alpha_{\mathrm{Above}}^{\mathrm{Put}(b,l)}(x,y)$ for the action $\mathrm{Put}(b,l)$ defined in Fig. 2 is the formula

$$(x=b\wedge y=l)\vee\exists z\big(x=b\wedge y=z\wedge\mathrm{Above}(l,z)\big). \tag{4.1}$$

The semantics of the Delete group is similar to the Add group, except in this case each clause corresponds to an individual delete condition $\hat{\delta}_R$ as given in Table 1. The disjunction of the individual $\hat{\delta}_R$'s defines the overall delete condition δ_R for relation R. If no delete conditions are specified for R, δ_R is taken to be the formula FALSE (i.e. nothing is to be deleted from R). For example, the overall delete condition $\delta_{\mathrm{Above}}^{\mathrm{Put}(b,l)}(x,y)$ for the action $\mathrm{Put}(b,l)$ is the formula

$$\exists z\big(x=b\wedge y=z\wedge z\neq l\wedge\neg\mathrm{Above}(l,z)\big). \tag{4.2}$$

Update groups are used to specify the update conditions for changing the values of function symbols. An Update group consists of a set of clauses of the forms shown in the left-hand column of Table 2. In this table, F is a function symbol, $\tau_1,\ldots,\tau_n,\tau$ are terms, ψ is a formula, $z_1,\ldots,z_k$ are variable symbols that appear in terms $\tau_1,\ldots,\tau_n,\tau$ but not in the parameter list of the action schema, and $x_1,\ldots,x_n$ are variable symbols that are distinct from the variables $z_1,\ldots,z_n$ and that do not appear in $\tau_1,\ldots,\tau_n,\psi$, or the parameter list of the action schema. The symbol C is a function of zero arguments and is included to illustrate this special case. Each clause in the left-hand column of Table 2 corresponds to the individual update condition $\hat{\mu}_F/\hat{\mu}_C$ defined in the right-hand column of Table 2. The disjunction of the individual update conditions defines the overall update condition μ_F/μ_C for the symbol F/C. If no update conditions are specified for F/C, then μ_F/μ_C is taken to be the formula FALSE (i.e. no updates take place

Clause	$\hat{\mu}_F(x_1,\ldots,x_n,y)/\hat{\mu}_C(y)$
$F(\tau_1,\ldots,\tau_n) \leftarrow \tau$	$(x_1 = \tau_1 \wedge \cdots \wedge x_n = \tau_n \wedge y = \tau)$
$F(\tau_1,\ldots,\tau_n) \leftarrow \tau$ if ψ	$(x_1 = \tau_1 \wedge \cdots \wedge x_n = \tau_n \wedge y = \tau \wedge \psi)$
$F(\tau_1,\ldots,\tau_n) \leftarrow \tau$ for all $z_1,\ldots,z_k$	$\exists z_1,\ldots,z_k\,(x_1 = \tau_1 \wedge \cdots \wedge x_n = \tau_n \wedge y = \tau)$
$F(\tau_1,\ldots,\tau_n) \leftarrow \tau$ for all $z_1,\ldots,z_k$ such that ψ	$\exists z_1,\ldots,z_k\,(x_1 = \tau_1 \wedge \cdots \wedge x_n = \tau_n \wedge y = \tau \wedge \psi)$
$C \leftarrow \tau$	$(y = \tau)$
$C \leftarrow \tau$ if ψ	$(y = \tau \wedge \psi)$
$C \leftarrow \tau$ for all $z_1,\ldots,z_k$	$\exists z_1,\ldots,z_k(y = \tau)$
$C \leftarrow \tau$ for all $z_1,\ldots,z_k$ such that ψ	$\exists z_1,\ldots,z_k(y = \tau \wedge \psi)$

TABLE 2. Update clauses and their meanings

for that symbol). For example, $\mu_{\text{Volume}}^{\text{Pour}(p,q)}(x,y)$ for the action $\text{Pour}(p,q)$ defined in Fig. 2 is the formula

$$\begin{aligned}&\big[x = q \wedge y = \min\big(\text{Capacity}(q), \text{Volume}(p) + \text{Volume}(q)\big)\big]\\ &\vee \big[x = p \wedge y = \max\big(0, \text{Volume}(p) - \text{Capacity}(q) + \text{Volume}(q)\big)\big].\end{aligned}$$

5 Reasoning about the effects of actions

Although there are many kinds of inferences one might want to make about actions and their effects, in automatic planning one is primarily interested in determining whether a particular action or sequence of actions is executable, and whether a given condition is true after executing the actions. The ADL language does not directly support such inferences; however, ADL schemas can be converted to another form that allows such reasoning to be performed using the deductive apparatus of first-order logic. As discussed below, more efficient reasoning methods can also be used in certain special cases.

Consider the problem of reasoning about the effects of actions from the point of view of the general state-transition framework. Suppose that we wish to determine the truth-value of the formula φ after executing action a. In general, we might only know that the world is initially in one of a given set of states I, but that the precise state is not known, except when I is a singleton set. Let $a(I)$ be the set of states that can potentially result if action a is successfully executed when the world is in one of these initial states; that is,

$$a(I) = \{t \mid \exists s\, \langle s,t\rangle \in a \text{ and } s \in I\}.$$

Then φ will be true after successfully executing action a if and only if $a(I) \models \varphi$. More generally, the set of states that can potentially result if the sequence of actions $a_1,\ldots,a_n$ is successfully executed is given by

$$a_n \circ \cdots \circ a_2 \circ a_1(I) = a_n(\cdots(a_2(a_1(I)))\cdots).$$

Thus, φ will be true after successfully executing actions $a_1,\ldots,a_n$ if and only if

$$a_n \circ \cdots \circ a_1(I) \models \varphi.$$

It is important to note that the definition of $a(I)$ does not presume that action a is executable from every state in the set I, it merely considers the possible effects of action a for those states in which a is executable; likewise for the sequence of actions $a_1, \ldots, a_n$. In order for action a to be executable from every state in I, it must be the case that

$$I \subseteq \mathrm{dom}(a) = \{s \mid \exists t \, \langle s, t \rangle \in a\}.$$

For the sequence $a_1, \ldots, a_n$ to be executable from every state in I, the first action a_1 must executable from every state in I and each subsequent action must executable from every state that can potentially result if the preceding actions are successfully executed. Thus, it must be the case that

$$\begin{aligned} I &\subseteq \mathrm{dom}(a_1) \\ a_{i-1} \circ \cdots \circ a_1(I) &\subseteq \mathrm{dom}(a_i), \qquad 1 < i \leq n. \end{aligned}$$

The set of initial states I is usually defined by a set of first-order formulas Γ such that

$$I = \{s \mid s \models \Gamma\}.$$

As discussed in Section 3, it is also typical for the states in which an action a can be executed to be defined by a first-order formula π^a such that

$$\mathrm{dom}(a) = \{s \mid \exists t \, \langle s, t \rangle \in a\} = \{s \mid s \models \pi^a\}.$$

To extend the notation to incorporate these conventions, let $a(\Gamma) = a\big(\{s \mid s \models \Gamma\}\big)$. It then follows that a sequence of actions $a_1, \ldots, a_n$ is executable from any initial state satisfying Γ if and only if

$$\begin{aligned} \Gamma &\models \pi^{a_1} \\ a_{i-1} \circ \cdots \circ a_1(\Gamma) &\models \pi^{a_i}, \qquad 1 < i \leq n. \end{aligned}$$

Because the first expression is expressed in terms of logical implication, the executability of action a_1 can be verified directly using first-order logic. Unfortunately, the second expression cannot be verified directly because $a_{i-1} \circ \cdots \circ a_1(\Gamma)$ is neither a formula nor a set of formulas, but a set of states. Likewise, first-order logic cannot be directly used to determine whether a given formula φ is true after successfully executing $a_1, \ldots, a_n$, since this is the case if and only if

$$a_n \circ \cdots \circ a_1(\Gamma) \models \varphi.$$

Additional means for reasoning about the effects of actions are therefore required.

For actions that can be described in ADL, the additional mechanism is provided by regression operators [75]. In the general case, a regression operator for an action is a function from formulas to formulas whose value indicates the conditions that must true prior to the execution of the action in order for the argument of the function to be true after execution. Stated formally,

DEFINITION 5.1
A *regression operator* for action a is a function a^{-1} from formulas to formulas such that, for every formula φ and every pair of states $\langle s, t \rangle \in a$, if $s \models a^{-1}(\varphi)$, then $t \models \varphi$.

Regression operators thus have the general property that if $\Gamma \models a^{-1}(\varphi)$, then $a(\Gamma) \models \varphi$. Likewise, by induction, if $\Gamma \models a_1^{-1} \circ \cdots \circ a_n^{-1}(\varphi)$, then $a_n \circ \cdots \circ a_1(\Gamma) \models \varphi$. Because the value of a regression operator is a formula, the proof mechanisms of first-order logic can be used to determine if $\Gamma \models a_1^{-1} \circ \cdots \circ a_n^{-1}(\varphi)$.

It is worth noting that Waldinger introduced the notion of a regression operator as the planning counterpart of what is known in program verification as predicate transformers for computing (weakest) sufficient preconditions for partial correctness [11, 28]. Partial correctness in this case means correctness assuming the program terminates; termination must be established separately in order to demonstrate total correctness. As discussed in Section 2, program termination corresponds to action executability in the state-transition model of action. Thus, as discussed above, if $\Gamma \models a^{-1}(\varphi)$, then φ will be true after executing action a in a state satisfying Γ, assuming that a is executable. Executability must be established separately by demonstrating that $\Gamma \models \pi^a$.

Note also that the definition of a regression operator given above requires only that $a^{-1}(\varphi)$ be a sufficient condition for φ to be true after action a is performed. It is therefore possible for there to be a pair of states $\langle s, t\rangle \in a$ such that $t \models \varphi$ even though $s \not\models a^{-1}(\varphi)$. Consequently, regression operators in and of themselves do not guarantee completeness when reasoning about the effects of actions. However, for actions that can be described in ADL, it is possible to construct *tidy* regression operators (after Pratt [63]) that do yield completeness.

DEFINITION 5.2
A regression operator a^{-1} for an action a is said to be *tidy* just in the case that $s \models a^{-1}(\varphi)$ if and only if $t \models \varphi$ for every formula φ and every pair of states $\langle s, t\rangle \in a$.

For tidy regression operators, if the sequence of actions $a_1, \ldots, a_n$ is executable from any state satisfying Γ, then $a_n \circ \cdots \circ a_1(\Gamma) \models \varphi$ if and only if $\Gamma \models a_1^{-1} \circ \cdots \circ a_n^{-1}(\varphi)$. Moreover, by induction, $a_1, \ldots, a_n$ is executable if and only if

$$\begin{aligned} \Gamma &\models \pi^{a_1} \\ \Gamma &\models a_1^{-1} \circ \cdots \circ a_{i-1}^{-1}(\pi^{a_i}), \qquad 1 < i \leq n. \end{aligned}$$

Tidy regression operators together with the deductive apparatus of first-order logic provide a sound and complete means for reasoning about the executability of actions and their resulting effects.

The remainder of this section shows how to construct tidy regression operators from ADL schemas. Before proceeding, however, a few side remarks should be made. The first is that, although regression operators enable the problem of reasoning about the effects of actions to be reduced to the problem of proving theorems in first-order logic, this does not means that general-purpose theorem provers have to be used. Depending on what is known about the initial state, it might be possible to use specialized techniques that are far more efficient. For example, it might be the case that the initial state is completely known in the sense that one can enumerate both the relevant objects that exist in the world and the initial values of all relevant functions and relations among those objects. In that case, the set of formulas Γ describing the initial state would be categorical [70, 8, 74] and the semantic structure that defines the initial state could be represented explicitly as a relational database. Theorem proving would then reduce to the evaluation of formulas in the initial state, an operation that can be performed relatively efficiently using query optimization techniques from relational database theory [73, 76]. Other constraints on what is known about the initial state would similarly lead to other specialized techniques for carrying out the theorem-proving tasks.

Another side remark is to note that the dual of a regression operator is a *progression operator*. A progression operator (after Rosenschein [67]) is a function that maps a formula that holds prior to the execution of an action to another formula that holds after execution:

DEFINITION 5.3
A *progression operator* for action a is a function a^{+1} from formulas to formulas such that $a(\gamma) \models a^{+1}(\gamma)$ for every formula γ.

Consequently, if $a^{+1}(\gamma) \models \varphi$, then $a(\gamma) \models \varphi$, and if $a_n^{+1} \circ \cdots \circ a_1^{+1}(\gamma) \models \varphi$, then $a_n \circ \cdots \circ a_1(\gamma) \models \varphi$. To establish that $a(\Gamma) \models \varphi$, where Γ is an infinite set of formulas, one must find a finite subset of formulas $\{\gamma_1, \ldots, \gamma_n\} \subseteq \Gamma$ such that $a^{+1}(\gamma_1 \wedge \cdots \wedge \gamma_n) \models \varphi$. If Γ is finite, as is often the case in planning problems, then the conjunction of all formulas in Γ would be constructed before applying the progression operators.

Progression operators are the planning counterpart of what is known in program verification as predicate transformers for computing (strongest) postconditions [15, 27]. However, unlike regression operators, progression operators appear to have been discovered independently in automatic planning by Fikes and Nilsson [14] in their development of the STRIPS operator language.

As with regression operators, there is also a notion of tidy progression operators (after Pratt [63]):

DEFINITION 5.4
A progression operator a^{+1} for an action a is said to be *tidy* just in the case that $a(\gamma) \models \varphi$ if and only if $a^{+1}(\gamma) \models \varphi$ for every pair of formulas γ and φ.

For tidy progression operators, it is the case that $a_n \circ \cdots \circ a_1(\gamma) \models \varphi$ if and only if $a_n^{+1} \circ \cdots \circ a_1^{+1}(\gamma) \models \varphi$. Unlike tidy regression operators, this property holds independent of whether or not the sequence of actions $a_1, \ldots, a_n$ is executable. Therefore, tidy progression operators likewise provide both a sound and complete means of reasoning about the effects of actions.

STRIPS operators are a special case of progression operators. Given a set of formulas Γ that describes what is true prior to executing an action, the STRIPS operator for that action will transform Γ into a new set of formulas that describe what is true after executing the action. In practice, only finite sets of formulas Γ are considered, and the transformations defined by STRIPS operators preserve finiteness. Thus, STRIPS operators can be seen as satisfying Definition 5.3 by interpreting a set of formulas in the STRIPS framework as being a representation for the conjunction of those formulas.

Interestingly, it is shown in Section 4 that there are actions that can be described in ADL for which tidy progression operators do not exist. In order to reflect all of the effects of such actions, the progression operator would have to output an infinite set of formulas. Thus, for all intents and purposes, progression operators do not generally provide a means for reasoning about the effects of actions described in ADL that is simultaneously sound, complete, and practical. This result may at first seem surprising, since progression operators are conceptually the inverse of regression operators and tidy regression operators can be constructed. The implication of the result is that, except in cases where tidy progression operators do exist, it is not possible to construct semantically equivalent STRIPS-like operators from ADL schemas despite the syntactic similarities between the two languages.

5.1 Constructing regression operators

The approach for deriving tidy regression operators from ADL schemas is motivated by the following construction. First, let us augment the language L for describing states of the world with an additional set of relation and function symbols such that one new symbol is introduced for each existing symbol. We are thereby creating an augmented language L' by adding a new n-ary relation symbol R' for each existing n-ary relation symbol R and a new n-ary function symbol F' for each existing n-ary function symbol F. The new symbols we will call primed, the old ones non-primed. Variable symbols will be considered as neither primed nor non-primed. The primed symbols will be used to describe the state of the world that exists after action a is performed, while the non-primed symbols will describe the state of the world before a is performed.

The next step in the construction is to axiomatize the relationship between the primed and non-primed symbols. To axiomatize the set of states in which an action a can be performed, the precondition of execution π^a of a is introduced as an axiom. The formula π^a can be used directly because it contains only non-primed symbols and, hence, describes a condition that must be true when a is performed. The effect of performing action a is axiomatized by making use of (3.1)–(3.4). These equations define how the interpretation of each symbol changes when an action is performed. From these equations and the preconditions of execution, the following axiom schemas are obtained:

$$\pi^a \tag{5.1}$$

$$\forall x_1, \ldots, x_n \left[R'(x_1, \ldots, x_n) \leftrightarrow \varphi_R^a(x_1, \ldots, x_n) \right] \tag{5.2}$$

$$\forall x_1, \ldots, x_n, y \left[F'(x_1, \ldots, x_n) = y \leftrightarrow \varphi_F^a(x_1, \ldots, x_n, y) \right], \tag{5.3}$$

where

$$\begin{aligned} \varphi_R^a(x_1, \ldots, x_n) &\equiv \alpha_R^a(x_1, \ldots, x_n) \vee \left(R(x_1, \ldots, x_n) \wedge \neg \delta_R^a(x_1, \ldots, x_n) \right) \\ \varphi_F^a(x_1, \ldots, x_n, y) &\equiv \mu_F^a(x_1, \ldots, x_n, y) \vee \left(F(x_1, \ldots, x_n) = y \right. \\ &\qquad \left. \wedge \neg \exists y \, \mu_F^a(x_1, \ldots, x_n, y) \right). \end{aligned}$$

Note that there is one instance of (5.2) for each non-primed n-ary relation symbol R, and one instance of (5.3) for each non-primed n-ary function symbol F.

Now consider the problem of computing the regression of a formula φ. This can be done using the construction presented above by first replacing all function and relation symbols in φ by their primed counterparts, thereby producing the formula φ'. Because φ' contains only primed symbols, it describes some condition that might hold after action a has been performed. Using the axioms defined above, φ' can be transformed into a logically equivalent formula ψ that contains only non-primed symbols. Because ψ is equivalent to φ' and contains only non-primed symbols, it expresses the necessary and sufficient preconditions that must exist before a is performed so that φ' (and, hence, φ) will be true afterward. Thus, ψ corresponds to the tidy regression of φ with respect to a. The transformation of φ' into ψ can be accomplished by a simple procedure in which the atomic formulas of φ' are replaced by equivalent formulas as given by (5.1)–(5.3). This procedure can then be modified to obtain a method for constructing regression operators.

The transformation of φ' is carried out in two steps. The first step is to convert φ' into an equivalent canonical form. Every atomic subformula of the canonical φ' will then either be of the form $R'(z_1, \ldots, z_n)$, $F'(z_1, \ldots, z_n) = w$, or $z_1 = z_2$, where $z_1, \ldots, z_n$, and w are variables. The canonical φ' is then transformed into the equivalent formula ψ by replacing the atomic

subformulas of the canonical φ' with equivalent formulas using a form of safe substitution. The equivalent formulas are obtained from (5.1)–(5.3). Thus, we replace all occurrences of

$$\begin{aligned} R'(z_1,\ldots,z_n) \quad &\text{with} \quad \alpha_R^a(z_1,\ldots,z_n) \vee \big(R(z_1,\ldots,z_n) \wedge \neg\delta_R^a(z_1,\ldots,z_n)\big) \\ F'(z_1,\ldots,z_n) = w \quad &\text{with} \quad \mu_F^a(z_1,\ldots,z_n,w) \vee \big(F(z_1,\ldots,z_n) = w \\ &\qquad\qquad \wedge \neg\exists w\, \mu_F^a(z_1,\ldots,z_n,w)\big). \end{aligned}$$

In performing these substitutions, quantified variables are renamed as needed to prevent any free variables in the α, δ, and μ formulas other than their parameters (i.e. $z_1,\ldots,z_n$, and w) from becoming bound. If this renaming is not done, an equivalent formula is not guaranteed.

To transform φ' into its canonical form, we make use of the following theorem: if $\lambda(z)$ is a formula containing the free variable z, and if z does not appear in the term τ, then $\lambda(\tau)$ is logically equivalent to $\exists z(\tau = z \wedge \lambda(z))$. The reason for this equivalence is that there is only one possible value for z such that $(\tau = z \wedge \lambda(z))$ is true, and that is the value of τ. This theorem permits us to replace any occurrence of

$$\begin{aligned} R'(\ldots,\tau,\ldots) \quad &\text{with} \quad \exists z\big(\tau = z \wedge R'(\ldots,z,\ldots)\big) \\ F'(\ldots,\tau,\ldots) = w \quad &\text{with} \quad \exists z\big(\tau = z \wedge F'(\ldots,z,\ldots) = w\big) \\ F'(\ldots) = \tau \quad &\text{with} \quad \exists z\big(\tau = z \wedge F'(\ldots) = z\big) \\ w = \tau \quad &\text{with} \quad \tau = w, \end{aligned}$$

where w is a variable, τ is a term that is not a variable, and z is a variable that does not appear in $R'(\ldots,\tau,\ldots)$, $F'(\ldots,\tau,\ldots) = w$, or $F'(\ldots) = \tau$, nor as a free variable in any of the α, δ, or μ formulas that define the effects of action a. This last requirement ensures that the free variables of the α, δ, or μ formulas do not become bound by the existential quantifiers introduced by the substitutions above. To put φ' in canonical form, the substitutions are repeatedly applied until no further substitutions are possible. The rule for replacing $w = \tau$ with $\tau = w$ is needed, since the other substitution rules require non-variable symbols to appear on the right-hand side of the equal sign.

The following equations define a class of regression operators based on the transformations described above. Other equation sets are also possible. The equations below may therefore be regarded as a minimally sufficient set. In the base case, $a^{-1}(\varphi)$ is given by

$$a^{-1}(\text{TRUE}) \equiv \text{TRUE} \tag{5.4}$$

$$a^{-1}(\text{FALSE}) \equiv \text{FALSE} \tag{5.5}$$

$$a^{-1}\big(R(z_1,\ldots,z_n)\big) \equiv \alpha_R^a(z_1,\ldots,z_n) \vee \big(R(z_1,\ldots,z_n) \wedge \neg\delta_R^a(z_1,\ldots,z_n)\big) \tag{5.6}$$

$$a^{-1}\big(F(z_1,\ldots,z_n) = w\big) \equiv \mu_F^a(z_1,\ldots,z_n,w) \vee \big(F(z_1,\ldots,z_n) = w \wedge \neg\exists w\, \mu_F^a(z_1,\ldots,z_n,w)\big) \tag{5.7}$$

$$a^{-1}(P) \equiv \alpha_P^a \vee (P \wedge \neg\delta_P^a) \tag{5.8}$$

$$a^{-1}(C = w) \equiv \mu_C^a(w) \vee \big(C = w \wedge \neg\exists w\, \mu_C^a(w)\big) \tag{5.9}$$

$$a^{-1}(z_1 = z_2) \equiv (z_1 = z_2), \tag{5.10}$$

where $z_1,\ldots,z_n$ and w are variable symbols. These equations correspond to replacing the atomic formulas of φ' with equivalent formulas. Note that (5.8) is a special case of (5.6), and (5.9) is a

special case of (5.7). They are included to illustrate the special cases of zero-place relations (i.e. proposition symbols) and zero-place functions (i.e. 'constant' symbols in first-order logic). The following equations transform atomic formulas into their canonical forms:

$$a^{-1}\big(R(\ldots,\tau,\ldots)\big) \equiv \exists z\big[a^{-1}(\tau = z) \wedge a^{-1}\big(R(\ldots,z,\ldots)\big)\big] \quad (5.11)$$

$$a^{-1}\big(F(\ldots,\tau,\ldots) = w\big) \equiv \exists z\big[a^{-1}(\tau = z) \wedge a^{-1}\big(F(\ldots,z,\ldots) = w\big)\big] \quad (5.12)$$

$$a^{-1}\big(F(\ldots) = \tau\big) \equiv \exists z\big[a^{-1}(\tau = z) \wedge a^{-1}\big(F(\ldots) = z\big)\big] \quad (5.13)$$

$$a^{-1}(C = \tau) \equiv \exists z\big[a^{-1}(\tau = z) \wedge a^{-1}(C = z)\big] \quad (5.14)$$

$$a^{-1}(w = \tau) \equiv a^{-1}(\tau = w), \quad (5.15)$$

where w is a variable symbol, τ is a term other than a variable symbol, and z is a variable symbol that does not appear in $R(\ldots,\tau,\ldots)$, $F(\ldots,\tau,\ldots) = w$, $F(\ldots) = \tau$, or $C = \tau$, nor as a free variable in any of the add, delete, or update (α, δ, or μ) formulas that define the effects of action a. Note that (5.14) is a special case of (5.13). The constraint that z not appear as a free variable in the add, delete, and update conditions was not included in earlier versions of the above equations [53, 54, 58]. The requirement as presented here corrects this omission. Finally, the following equations allow (5.4)–(5.15) to be applied to all atomic subformulas of a formula:

$$a^{-1}(\neg\varphi) \equiv \neg a^{-1}(\varphi) \quad (5.16)$$

$$a^{-1}(\varphi \wedge \psi) \equiv a^{-1}(\varphi) \wedge a^{-1}(\psi) \quad (5.17)$$

$$a^{-1}(\varphi \vee \psi) \equiv a^{-1}(\varphi) \vee a^{-1}(\psi) \quad (5.18)$$

$$a^{-1}(\varphi \rightarrow \psi) \equiv a^{-1}(\varphi) \rightarrow a^{-1}(\psi) \quad (5.19)$$

$$a^{-1}(\varphi \leftrightarrow \psi) \equiv a^{-1}(\varphi) \leftrightarrow a^{-1}(\psi) \quad (5.20)$$

$$a^{-1}(\forall x\,\varphi) \equiv \forall z\, a^{-1}([z/x]\varphi) \quad (5.21)$$

$$a^{-1}(\exists x\,\varphi) \equiv \exists z\, a^{-1}([z/x]\varphi), \quad (5.22)$$

where z is a variable symbol (possibly x itself) that does not appear as a free variable in either $\forall x\,\varphi$ or $\exists x\,\varphi$, nor in any of the α, δ, or μ formulas that define the effects of action a. The safe substitution of z for x in (5.21) and (5.22) ensures that the free variables of the α, δ, or μ formulas introduced by (5.6)–(5.9) do not become accidentally bound. Note that if x does not appear as a free variable in these formulas, then z can be x itself, in which case no renaming of variables occurs. This renaming of bound variables was not included in earlier versions of the above equations [53, 54, 58], making the earlier equations applicable only when the renaming of x is not required. Equations (5.21) and (5.22), on the other hand, are completely general.

THEOREM 5.5
For every action a that can be represented in ADL, (5.4)–(5.22) define a tidy regression operator for that action.

The proof of Theorem 5.5 appears in the appendix. The proof, however, does not rely on (5.1)–(5.3) that were introduced to motivate the equations. Instead, it is based purely on the semantics of states and the semantics of ADL presented in Sections 2 and 3. However, the construction defined by (5.1)–(5.3) is interesting in and of itself because it provides an alternate characterization of the semantics of ADL.

THEOREM 5.6
For any action a that can be represented in ADL and any pair of states s and t, it is the case that $\langle s,t\rangle \in a$ if and only if the corresponding semantic structure for language L' satisfies (5.1)–(5.3).

A formal statement and proof of Theorem 5.6 appears in the appendix.

To illustrate the use of the regression equations, consider the plan of executing $\mathrm{Put}(B, C)$ followed by $\mathrm{Put}(A, B)$ (i.e. put B on top of C then A on top of B). Suppose that we wish to determine whether this plan achieves $\mathrm{Above}(A, D)$. This is accomplished by computing $\mathrm{Put}(B, C)^{-1}\left[\mathrm{Put}(A, B)^{-1}\left(\mathrm{Above}(A, D)\right)\right]$ and then comparing the result with what is known about the initial state. From (5.4)–(5.22) and (4.1) and (4.2),

$$\begin{aligned}
&\mathrm{Put}(A, B)^{-1}\left(\mathrm{Above}(A, D)\right) \\
&\quad\equiv\ (A = A \wedge D = B) \\
&\qquad \vee \exists z\left(A = A \wedge D = z \wedge \mathrm{Above}(B, z)\right) \\
&\qquad \vee \left[\mathrm{Above}(A, D) \wedge \neg\exists z\left(A = A \wedge D = z \wedge z \neq B \wedge \neg\mathrm{Above}(B, z)\right)\right],
\end{aligned}$$

which simplifies to

$$(D = B) \vee \mathrm{Above}(B, D).$$

Applying $\mathrm{Put}(B, C)^{-1}$ to this result yields

$$\begin{aligned}
&\mathrm{Put}(B, C)^{-1}\left[(D = B) \vee \mathrm{Above}(B, D)\right] \\
&\quad\equiv\ \exists z\left[\left(\mathrm{FALSE} \vee (D = z \wedge \neg\exists z\,\mathrm{FALSE})\right) \wedge \left(\mathrm{FALSE} \vee (B = z \wedge \neg\exists z\,\mathrm{FALSE})\right)\right] \\
&\qquad \vee \left[\mathrm{Above}(B, D) \wedge \neg\exists z\left(B = B \wedge D = z \wedge z \neq C \wedge \neg\mathrm{Above}(C, z)\right)\right] \\
&\qquad \vee (B = B \wedge D = C) \vee \exists z\left(B = B \wedge D = z \wedge \mathrm{Above}(C, z)\right),
\end{aligned}$$

which simplifies to

$$(D = B) \vee (D = C) \vee \mathrm{Above}(C, D).$$

Thus, A will be above D after executing the plan if and only if C is above D in the initial state, or D is either B or C.

6 Properties of regression operators

A general method has been proposed [53, 54, 56, 57, 59] for solving planning problems in which the effects of the actions are context-dependent; that is, in cases where the changes that take place as the result of an action can depend on what is true at the time the action is performed. The method employs regression operators as an integral part of the planning process. To use the method, one must first construct regression operators for the actions that can appear in a plan. As shown in the previous section, tidy regression operators are readily constructed from ADL schemas, making ADL suitable for use with the planning method. However, other languages can also be used provided that they permit suitable regression operators to be constructed. This section presents a number of theorems relating to the kinds of regression operators that can be constructed for an action. The proofs of the theorems appear in the appendix.

One of the interesting things to note about the regression operators defined by (5.4)–(5.22) is that $a^{-1}(\neg\varphi)$ is equal to $\neg a^{-1}(\varphi)$. As it turns out, any regression operator that has this property must necessarily be tidy. This fact is interesting, as it provides a simple way of characterizing when a regression operator is tidy.

THEOREM 6.1

A regression operator a^{-1} for an action a is tidy if and only if, for every formula φ, $a^{-1}(\neg\varphi)$ and $\neg a^{-1}(\varphi)$ have the same truth value in every state in which action a can be executed (i.e., for every formula φ and every pair of states $\langle s, t\rangle \in a$, $s \models a^{-1}(\neg\varphi)$ if and only if $s \models \neg a^{-1}(\varphi)$).

Actions with tidy regression operators have an additional property, which is that they are *quasi-deterministic*. As discussed in Section 2, an action is deterministic if and only if there is a unique succedent state for every state in which the action can be performed. However, another notion of determinism arises when we consider only the formulas that are true in a state. Since formulas are finite objects, there can be only countably many of them. However, because the domains of states can be infinite, there can be uncountably many states. Consequently, it is possible for two states to be distinct and yet be indistinguishable with respect to the formulas that are true in those states. This observation leads to a weaker notion of determinism which we will call quasi-determinism.

DEFINITION 6.2
Two states s and s' are said to be *elementary equivalent* [70, 8, 74] just in the case that $s \models \varphi$ if and only if $s' \models \varphi$ for every formula φ.

DEFINITION 6.3
An action a is said to be *quasi-deterministic* just in the case that, for every pair of states $\langle s, t \rangle \in a$ and every pair of states $\langle s', t' \rangle \in a$, if s and s' are elementary equivalent, then t and t' are elementary equivalent.

Thus, if we divide the states of the world into equivalence classes based on what is true in those states, the execution of a quasi-deterministic action will cause the world to transition from a state in one equivalence class to a state in a unique succedent equivalence class. It is in this sense that an action can behave deterministically for all intents and purposes without being strictly deterministic. Given the above definitions, the following theorem holds:

THEOREM 6.4
If an action has a tidy regression operator, then that action is quasi-deterministic.

If a tidy regression operator cannot be constructed for an action, one should at least attempt to construct a regression operator that has certain other desirable properties. For example, if φ and φ' are logically equivalent formulas, then $a^{-1}(\varphi)$ and $a^{-1}(\varphi')$ should likewise be logically equivalent. More generally, if $\varphi \models \psi$, it should also be the case that $a^{-1}(\varphi) \models a^{-1}(\psi)$. This second requirement includes the first, since φ and φ' are equivalent if and only if $\varphi \models \varphi'$ and $\varphi' \models \varphi$. The rationale behind the requirement is that if φ logically implies ψ, then by achieving φ we would automatically achieve ψ. Hence, if $a^{-1}(\varphi)$ is true, we should expect $a^{-1}(\psi)$ to be true as well, since both φ and ψ would be true after performing action a.

Another desirable property arises for conjunctions of formulas. If both $a^{-1}(\varphi)$ and $a^{-1}(\psi)$ are true when action a is performed, $\varphi \wedge \psi$ will be true immediately afterward. Consequently, if the formula $a^{-1}(\varphi) \wedge a^{-1}(\psi)$ is true, it is reasonable to expect $a^{-1}(\varphi \wedge \psi)$ to be true as well. These considerations give rise to the following definition:

DEFINITION 6.5
A regression operator a^{-1} for an action a is said to be *coherent* if and only if the following holds for ever pair of formulas φ and ψ:

1. If $\varphi \models \psi$, then $a^{-1}(\varphi) \models a^{-1}(\psi)$.
2. $a^{-1}(\varphi) \wedge a^{-1}(\psi) \models a^{-1}(\varphi \wedge \psi)$.

Note that, since $\varphi \wedge \psi \models \varphi$ and $\varphi \wedge \psi \models \psi$, the first clause of Definition 6.5 also implies that $a^{-1}(\varphi \wedge \psi) \models a^{-1}(\varphi) \wedge a^{-1}(\psi)$ for every pair of formulas φ and ψ. The two clauses combined therefore imply that $a^{-1}(\varphi \wedge \psi)$ and $a^{-1}(\varphi) \wedge a^{-1}(\psi)$ must be logically equivalent. Hence, when a coherent regression operator is applied to a set of facts, the result is effectively

independent of how those facts are expressed, and whether they are expressed separately or in conjunction. This independence is highly desirable, since it eliminates any potential dependence on syntax. Coherent regression operators should therefore be constructed whenever possible.

Theorem 5.5 in the previous section demonstrates that (5.4)–(5.22) define tidy regression operators for actions representable in ADL. The following theorem shows that these regression operators are also coherent.

THEOREM 6.6
If a is an action described in ADL, and if a^{-1} is the regression operator defined by (5.4)–(5.22), then a^{-1} is a coherent regression operator for action a.

7 ADL and progression operators

The use of regression operators to reason about the effects of actions contrasts with most previous work in automatic planning in which progression operators based on the STRIPS language were used. The reason that progression operators are not employed here is that they do not provide a reasoning mechanism that is simultaneously sound, complete, and practical for all actions representable in ADL.

As discussed in Section 5, a progression operator a^{+1} for an action a is a function from formulas to formulas such that $a(\gamma) \models a^{+1}(\gamma)$ for every formula γ. Progression operators are used to 'progress' the description of an initial state forward through a plan to determine the conditions that would hold at various points in the plan. The formula given by $a^{+1}(\gamma)$ is called a postcondition of γ with respect to action a:

DEFINITION 7.1
A formula φ is said to be a *postcondition* of a formula γ with respect to an action a if and only if $a(\gamma) \models \varphi$ (i.e. for every pair of states $\langle s, t\rangle \in a$, if $s \models \gamma$, then $t \models \varphi$).

It is desirable for $a^{+1}(\gamma)$ to completely describe what is true after action a has been performed given that γ was true beforehand. A progression operator that has this property is said to be tidy (Definition 5.4) and the formula given by $a^{+1}(\gamma)$ is said to be a strongest postcondition of γ with respect to a:

DEFINITION 7.2
A formula φ is a *strongest postcondition* of a formula γ with respect to an action a if and only if φ is a postcondition of γ with respect to a and $\varphi \models \psi$ for every other postcondition ψ.

Note that, if φ is a strongest postcondition of γ, then $a(\gamma) \models \psi$ if and only if $\varphi \models \psi$. Hence, a^{+1} is a tidy progression operator if and only if $a^{+1}(\gamma)$ is a strongest postcondition of γ for all γ. Oddly enough, the following theorem holds:

THEOREM 7.3
There exist actions representable in ADL that do not have tidy progression operators.

Theorem 7.3 is surprising, since a progression operator is conceptually the inverse of a regression operator, and all actions that can be represented in ADL have tidy regression operators. The proof that appears in the appendix exhibits an action a and a formula γ with the property that, for every postcondition of γ, there exists another postcondition that is strictly stronger. Because there is no single strongest postcondition, an infinite set of formulas would have to be employed to fully represent all of the postconditions of γ. Infinite sets of formulas pose many problems from the standpoint of implementing a planning system. In practice, they are best avoided whenever

possible. Thus, compared to regression operators, progression operators do not provide a means of reasoning about the effects of all actions that can be represented in ADL that is simultaneously sound, complete, and practical.

8 ADL and the situation calculus

The situation calculus is a discipline for constructing first-order theories about the effects of actions. To use the calculus, an additional argument is added to every relation and function whose value can change when an action is performed. The additional argument ranges over states, which are called situations in the situation calculus. This additional argument enables one to refer to the values of relations and functions in different states. For example, the relation $\mathrm{On}(x, y)$ would become $\mathrm{On}(x, y, s)$, and the function $\mathrm{Volume}(x)$ would become $\mathrm{Volume}(x, s)$. This is likewise done for functions with zero arguments. Thus, the zero-place function *LastBlockMoved* would become $\mathit{LastBlockMoved}(s)$. An additional function $\mathrm{result}(a, s)$ is also introduced into the language that maps actions and situations to situations. This function is used to encode the state transitions for each action.

Once situational arguments have been added, axioms can be written that describe the effects of actions in different situations. The axioms typically have one of two forms. Axioms of the first kind are called effect axioms and describe the changes that take place when an action is performed. These axioms have the form

$$\forall s\left[\pi^a(s) \wedge \varphi_1(s) \rightarrow \psi_1(\mathrm{result}(a, s))\right],$$

or equivalently

$$\forall s\left[\pi^a(s) \rightarrow \left(\varphi_1(s) \rightarrow \psi_1(\mathrm{result}(a, s))\right)\right].$$

In the above axiom schema, s ranges over situations, π^a describes the preconditions for the execution of action a, ψ_1 describes a condition that becomes true as a result of performing a, and φ_1 describes the conditions under which this change takes place. There may also be some additional quantified variables other than the situation variable s, depending on the nature of the formulas π^a, ψ_1, and φ_1, and on whether a is a parameterized action.

Axioms of the second kind are called frame axioms and describe properties of the world that are not affected by an action. These axioms have the form

$$\forall s\left[\pi^a(s) \wedge \psi_2(s) \wedge \varphi_2(s) \rightarrow \psi_2(\mathrm{result}(a, s))\right],$$

or equivalently

$$\forall s\left[\pi^a(s) \rightarrow \left(\psi_2(s) \wedge \varphi_2(s) \rightarrow \psi_2(\mathrm{result}(a, s))\right)\right],$$

where $\psi_2(\mathrm{result}(a, s))$ is the formula obtained by performing a safe substitution of $\mathrm{result}(a, s)$ for s in $\psi_2(s)$. The formula ψ_2 describes a condition that remains unaltered when action a is performed, while φ_2 describes the circumstances under which this occurs. As before, π^a describes the preconditions for the execution of a.

Two things should be noted about this way of modelling actions as compared to the state-transition model defined in Section 2. The first is that, since $\mathrm{result}(a, s)$ is a function, all actions are presumed to be deterministic. An effect axiom of the form $\forall s\left[\pi^a(s) \wedge \varphi_1(s) \rightarrow \left(\psi_1(\mathrm{result}(a, s)) \vee \psi_2(\mathrm{result}(a, s))\right)\right]$ might at first appear as if it is defining a non-deterministic effect, but in fact it is merely saying that the effect is not completely known. Executing action a several times from the same state s would always produce the same effect, it is just that we

only have partial knowledge of the exact effect. This is different from the definition of non-determinism presented in Section 2 in which an action can have different effects each time it is executed from the same state.

The second thing to note is that, because the semantics of first-order logic require functions for be defined everywhere, there always exists a state t such that $t = \text{result}(a, s)$ for every state s. Thus, from the point of view of semantics, the situation calculus actually allows actions to be executed in every state. However, by including the intended preconditions of execution as part of the antecedent in the effect axioms and frame axioms, no conclusions can be drawn when an action or sequence of actions is not intended to be executable. The reason is that the axioms do not place any constraints on the state transitions that can occur when an action is performed in a state that does not satisfy the intended preconditions of execution. Hence, the effects of the actions in those states is completely unknown. Executability is thus being factored in by restricting the set of theorems that can be logically inferred.

Keeping these caveats in mind, it is possible to construct an equivalent axiomatization of ADL schemas in the situation calculus. In particular, (5.1)–(5.3) can be adapted almost directly, yielding the following axioms for every n-ary relation symbol R and every n-ary function symbol F:

$$\forall x_1, \ldots, x_n, s\big[\pi^a(s) \rightarrow \big(R(x_1, \ldots, x_n, \text{result}(a, s)) \leftrightarrow \varphi_R^a(x_1, \ldots, x_n, s)\big)\big]$$
$$\forall x_1, \ldots, x_n, y, s\big[\pi^a(s) \rightarrow \big(F(x_1, \ldots, x_n, \text{result}(a, s)) = y \leftrightarrow \varphi_F^a(x_1, \ldots, x_n, y, s)\big)\big],$$

where

$$\begin{aligned}
\varphi_R^a(x_1, \ldots, x_n, s) \quad &\equiv \quad \alpha_R^a(x_1, \ldots, x_n, s) \vee \big(R(x_1, \ldots, x_n, s) \\
&\qquad\qquad \wedge \neg\delta_R^a(x_1, \ldots, x_n, s)\big) \\
\varphi_F^a(x_1, \ldots, x_n, y, s) \quad &\equiv \quad \mu_F^a(x_1, \ldots, x_n, y, s) \vee \big(F(x_1, \ldots, x_n, s) = y \\
&\qquad\qquad \wedge \neg\exists y\, \mu_F^a(x_1, \ldots, x_n, y, s)\big),
\end{aligned}$$

and where $\alpha_R^a(x_1, \ldots, x_n, s)$, $\delta_R^a(x_1, \ldots, x_n, s)$, and $\mu_F^a(x_1, \ldots, x_n, y, s)$ are constructed by introducing a newly-created situational variable s as an extra argument to every function and relation that appears in the corresponding formulas obtained from the ADL schema of action a. These axioms decompose into a logically equivalent set of axioms consisting of the effect axioms

$$\forall x_1, \ldots, x_n, s\big[\pi^a(s) \wedge \alpha_R^a(x_1, \ldots, x_n, s) \rightarrow R(x_1, \ldots, x_n, \text{result}(a, s))\big] \quad (8.1)$$

$$\forall x_1, \ldots, x_n, s\big[\pi^a(s) \wedge \delta_R^a(x_1, \ldots, x_n, s) \wedge \neg\alpha_R^a(x_1, \ldots, x_n, s) \rightarrow \neg R(x_1, \ldots, x_n, \text{result}(a, s))\big] \quad (8.2)$$

$$\forall x_1, \ldots, x_n, y, s\big[\pi^a(s) \wedge \mu_F^a(x_1, \ldots, x_n, y, s) \rightarrow F(x_1, \ldots, x_n, \text{result}(a, s)) = y\big] \quad (8.3)$$

and the frame axioms

$$\forall x_1, \ldots, x_n, s\big[\pi^a(s) \wedge R(x_1, \ldots, x_n, s) \wedge \neg\delta_R^a(x_1, \ldots, x_n, s) \rightarrow R(x_1, \ldots, x_n, \text{result}(a, s))\big] \quad (8.4)$$

$$\forall x_1, \ldots, x_n, s \big[\pi^a(s) \wedge \neg R(x_1, \ldots, x_n, s) \wedge \neg \alpha_R^a(x_1, \ldots, x_n, s) \rightarrow \neg R(x_1, \ldots, x_n, \text{result}(a, s))\big] \quad (8.5)$$

$$\forall x_1, \ldots, x_n, y, s \big[\pi^a(s) \wedge F(x_1, \ldots, x_n, s) = y \wedge \neg \exists y\, \mu_F^a(x_1, \ldots, x_n, y, s) \rightarrow F(x_1, \ldots, x_n, \text{result}(a, s)) = y\big]. \quad (8.6)$$

If the ADL schema is consistent, as defined in Definition 3.2, then the add and delete conditions α_R^a and δ_R^a are never true simultaneously in any state in which action a can be executed; that is,

$$\forall x_1, \ldots, x_n, s \big[\pi^a(s) \rightarrow \big(\neg \delta_R^a(x_1, \ldots, x_n, s) \wedge \neg \alpha_R^a(x_1, \ldots, x_n, s)\big)\big].$$

In that case, (8.2) simplifies to

$$\forall x_1, \ldots, x_n, s \big[\pi^a(s) \wedge \delta_R^a(x_1, \ldots, x_n, s) \rightarrow \neg R(x_1, \ldots, x_n, \text{result}(a, s))\big]. \quad (8.7)$$

Note that only one effect axiom and one frame axiom is needed in the case of function symbols, since functions are defined everywhere and there is always a unique y such that $F(x_1, \ldots, x_n, s) = y$.

Using the axioms schemas presented above, any ADL schema can be translated into an equivalent set of axioms in the situation calculus. In particular, the actions defined in Fig. 2 can be axiomatized, including the action Adjust for adjusting the thrust and attitude of a lunar lander. Thus, dynamic worlds and continuous processes can be modelled in the situation calculus using the approach described in Section 2, just as they can using ADL.

Based on (8.1)–(8.7), it is possible to define a restricted from of the situation calculus such that any ADL schema can be transformed into an equivalent axiomatization in this restricted form, and vice versa. In addition, the restricted form allows frame axioms to be derived from the effect axioms in a straight-forward manner. The problem of automatically deriving frame axioms from effect axioms is one version of the frame problem [45, 25, 9]. Traditional approaches to addressing this problem for the situation calculus are based on the technique of circumscription [46, 22, 23, 31, 36, 37, 38, 39, 71, 2, 3]. Circumscriptive approaches attempt to infer frame axioms essentially by inferring what is true in the 'smallest' semantic structures satisfying the effect axioms. Different definitions of 'smallest' lead to different forms of circumscription. The semantics of ADL, on the other hand, provides a solution to the frame problem that does not require the use of circumscription. It also avoids the issue of how to apply circumscription properly so as not to introduce unrealistic or erroneous frame axioms [22, 23, 31, 71, 2, 3, 39]. By constructing a restricted form of the situation calculus that is equivalent to ADL, these benefits can likewise be shared by the situation calculus in cases where the restricted form can be adhered to when axiomatizing problems of interest.

The only axioms that are allowed in the restricted form are effect axioms of the forms

$$\forall x_1, \ldots, x_n, s \big[\pi^a(s) \wedge \hat{\alpha}_R^a(x_1, \ldots, x_n, s) \rightarrow R(x_1, \ldots, x_n, \text{result}(a, s))\big] \quad (8.8)$$

$$\forall x_1, \ldots, x_n, s \big[\pi^a(s) \wedge \hat{\delta}_R^a(x_1, \ldots, x_n, s) \rightarrow \neg R(x_1, \ldots, x_n, \text{result}(a, s))\big] \quad (8.9)$$

$$\forall x_1, \ldots, x_n, y, s \big[\pi^a(s) \wedge \hat{\mu}_F^a(x_1, \ldots, x_n, y, s) \rightarrow F(x_1, \ldots, x_n, \text{result}(a, s)) = y\big], \quad (8.10)$$

where $x_1, \ldots, x_n, y, s$ are free variables of $\hat{\alpha}_R^a$, $\hat{\delta}_R^a$, and $\hat{\mu}_F^a$. Axioms of these forms correspond

to ADL clauses of the forms

$$\text{Add: } R(x_1, \ldots, x_n) \text{ for all } x_1, \ldots, x_n \text{ such that } \hat{\alpha}_R^a(x_1, \ldots, x_n)$$

$$\text{Delete: } R(x_1, \ldots, x_n) \text{ for all } x_1, \ldots, x_n \text{ such that } \hat{\delta}_R^a, (x_1, \ldots, x_n)$$

$$\text{Update: } F(x_1, \ldots, x_n) \leftarrow y \text{ for all } x_1, \ldots, x_n \text{ such that } \hat{\mu}_F^a(x_1, \ldots, x_n, y)$$

where the situational variable s has been removed from $\hat{\alpha}_R^a$, $\hat{\delta}_R^a$, and $\hat{\mu}_F^a$. As with ADL clauses, there can be several effect axioms for each n-ary relation symbol R or n-ary function symbol F.

Given a set of effect axioms Γ of the forms defined above, a set of frame axioms $\mathcal{F}(\Gamma)$ is then constructed according to (8.1)–(8.7). The overall add, delete, and update conditions α_R^a, δ_R^a, and μ_F^a that appear in (8.1)–(8.7) are constructed from the individual add, delete, and update conditions $\hat{\alpha}_R^a$, $\hat{\delta}_R^a$, and $\hat{\mu}_F^a$ that appear in (8.8)–(8.10). The construction is identical to that used for the individual add, delete, and update conditions constructed from ADL clauses as discussed in Section 4. Thus, if there are one or more individual add conditions $\hat{\alpha}_{R,1}^a, \ldots, \hat{\alpha}_{R,k}^a$ for a relation symbol R, the overall add conditions α_R^a is the disjunction of the individual add conditions:

$$\alpha_R^a(x_1, \ldots, x_n, s) \equiv \hat{\alpha}_{R,1}^a(x_1, \ldots, x_n, s) \wedge \cdots \wedge \hat{\alpha}_{R,k}^a(x_1, \ldots, x_n, s).$$

If there are no individual add conditions $\hat{\alpha}_R^a$ for relation symbol R in the set Γ, then α_R^a is taken to be the formula FALSE. Likewise, the overall delete conditions δ_R^a for a relation symbol R is the disjunction of the individual delete conditions $\hat{\delta}_{R,1}^a, \ldots, \hat{\delta}_{R,k}^a$, or the formula FALSE if there are no individual delete conditions for R in Γ. The overall update conditions μ_F^a for a function symbol F is the disjunction of the individual update conditions $\hat{\mu}_{F,1}^a, \ldots, \hat{\mu}_{F,k}^a$, or the formula FALSE if there are no individual update conditions for F in Γ. The resulting add, delete, and update conditions α_R^a, δ_R^a, and μ_F^a are then equivalent to those that appear in the ADL schema for the action. They are therefore inserted into (8.4)–(8.6) to generate the set of frame axioms $\mathcal{F}(\Gamma)$. The set of formulas $\Gamma \cup \mathcal{F}(\Gamma)$ then provides a complete axiomatization of the actions equivalent to the corresponding ADL schemas.

It should be noted that one could generate a new set of effect axioms Γ' by inserting the overall add, delete, and update conditions α_R^a, δ_R^a, and μ_F^a into (8.1)–(8.3) and (8.7). However, there would be no point to this, since the resulting set Γ' would be logically equivalent to Γ. The reason is that a collection of axioms of the form

$$\begin{gathered} \forall x_1, \ldots, x_n, s\left[\pi \wedge \varphi_1 \rightarrow \psi\right] \\ \vdots \\ \forall x_1, \ldots, x_n, s\left[\pi \wedge \varphi_n \rightarrow \psi\right] \end{gathered}$$

is equivalent to the single axiom

$$\forall x_1, \ldots, x_n, s\left[\pi \wedge \left(\varphi_1 \vee \cdots \vee \varphi_n\right) \rightarrow \psi\right],$$

and an axiom of the form $\forall x_1, \ldots, x_n, s\left[\pi \wedge \text{FALSE} \rightarrow \psi\right]$ is a tautology.

Note also that it would be pointless to introduce an effect axiom for a relation or function symbol if its interpretation is not affected by an action. The corresponding add, delete, or update condition would be the formula FALSE, making the corresponding effect axiom a tautology. Moreover, the fact that the add, delete, or update condition is FALSE would be automatically inferred in the process of constructing the frame axioms. Thus, Reiter's claim [65] regarding

the number of effect axioms that are needed to axiomatize the effects of actions in this restricted form is incorrect. Reiter claims that there must necessarily be $2 \cdot |\mathcal{A}| \cdot |\mathcal{F}|$ axioms in Γ, where $|\mathcal{A}|$ is the number of actions and $|\mathcal{F}|$ is the number of *fluents* (i.e. the number of symbols whose interpretations can change by performing actions). This claim does happen to be true assuming one includes tautological axioms in which the add, delete, and update conditions are the formula FALSE. However, if these tautologies are omitted, considerably shorter axiomatizations are obtained in cases where each action affects the interpretations of relatively few symbols. Reiter also makes several other claims, but these will be considered elsewhere.

9 Conclusions

ADL addresses an important issue in knowledge representation and automatic planning, which is to balance the expressiveness of a logical formalism against the computational cost of reasoning with that formalism. The expressive power of ADL approaches that of the situation calculus, yet its computational costs compare favourably to those of the STRIPS language. In particular, when the initial state of a planning problem is completely known, the initial state can be represented explicitly and query techniques developed for relational database systems can be used to perform reasoning tasks. Database techniques are not appropriate, however, when the initial state is only partially known. In such instances, a more general form of theorem proving must be used. This increases computational costs; however, the increase is inevitable and will necessarily be present in any representational language [34, 35].

One of the features of ADL is that it incorporates a solution to the frame problem that is the model-theoretic counterpart to the solution employed in the STRIPS operator language. As with STRIPS operators, the transformations that take place when an action is performed are represented as set differences. Consequently, one need only describe the changes that take place when an action is performed, not what remains unaltered. Unlike STRIPS operators, however, ADL schemas define transformations on the model-theoretic structures that underlie the semantics of first-order logic, whereas STRIPS operators define transformations on formulas. ADL thus has a true model-theoretic semantics. Moreover, the transformations in ADL can be functions of the situations in which the actions are performed, allowing actions with context-dependent effects to be represented as in the case of the lunar-landing example presented in Section 2. Such actions cannot be represented using STRIPS operators.

The solution to the frame problem embodied in ADL has been transferred to the situation calculus, yielding a restricted form of the situation calculus that is effectively equivalent to ADL in that any ADL schema can be translated into an equivalent axiomatization in this restricted form, and vice versa. In the restricted form, frame axioms are automatically derived from effect axioms in a manner that reflects the frame assumption embodied in ADL. By incorporating ADL's solution to the frame problem, the restricted form of the situation calculus avoids explicit use of circumscription [46, 22, 23, 31, 36, 37, 38, 39, 71, 2, 3].

To reason about the effects of actions represented in ADL, one can either employ the restricted form of the situation calculus described above or one can construct regression operators for this purpose. A method has been presented for constructing regression operators from ADL schemas that give the necessary and sufficient conditions that must exist prior to the execution of an action or sequence of actions in order for some desired condition to be true after execution. When combined with the theorem-proving and database-query techniques discussed above, regression operators provide a sound and complete means for reasoning about the effects of actions representable in ADL.

Remarkably, comparable progression operators cannot in general be constructed from ADL schemas. It has been shown that there are cases in which the progression operator for an ADL schema would have to produce an infinite set of formulas in order to fully reflect all of the effects of the corresponding action. The regression operators constructed from ADL schemas, on the other hand, always preserve finiteness. This result is surprising, since progression operators are conceptually the inverse of regression operators. The implication is that progression operators do not always provide a means for reasoning about the effects of actions representable in ADL that is simultaneously sound, complete, and practical relative to regression operators.

ADL was originally developed in conjunction with a planning technique suitable for actions with context-dependent effects [53, 54, 56, 57, 59]. This planning technique is actually independent of ADL and requires only that regression operators be provided for each action. ADL complements the technique by allowing the necessary regression operators to be readily constructed from descriptions of the effects of actions. As a separate formalism, ADL has been found to have desirable computational properties and to be useful in modelling a wide range of actions, including some that involve dynamic processes and the passage of time. These features, together with an associated planning technique, have already made ADL a useful adjunct to the representational techniques previously available in automatic planning. Indeed, a number of planning systems have already been developed that employ ADL-based representations [49, 10, 60, 64, 12, 32, 33]. In addition, the restricted form of the situation calculus derived from ADL has found use in applications other than planning [66].

Acknowledgements

I would like to thank Bob Moore for urging me to develop my planning ideas within a STRIPS-like framework, and Gio Wiederhold for introducing me to relational database theory. Their combined influence lead me to re-examine the STRIPS framework and develop the ADL language. I would also like to thank the reviewers and Marla and Corinne Babcock for their comments on earlier drafts of this paper.

References

[1] W. Ackermann. *Solvable Cases of the Decision Problem*. North-Holland, Amsterdam, 1954.

[2] A. B. Baker. A simple solution to the Yale shooting problem. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, pp. 11–20. Morgan Kaufmann, San Mateo, CA, 1989.

[3] A. B. Baker. Nonmonotonic reasoning in the framework of situation calculus. *Artificial Intelligence*, 49, 5–23, 1991.

[4] C. Baral and M. Gelfond. Representing concurrent actions in extended logic programming. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, pp. 866–871. Morgan Kaufmann, San Mateo, CA, 1993.

[5] M. Bauer, S. Biundo, D. Dengler, J. Keohler, and G. Paul. PHI—a logic-based tool for intelligent help systems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, pp. 460–466. Morgan Kaufmann, San Mateo, CA, 1993.

[6] P. Belegrinos and M. Georgeff. A model of events and processes. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pp. 506–511. Morgan Kaufmann, San Mateo, CA, 1991.

[7] S. Biundo, D. Dengler, and J. Koehler. Deductive planning and plan reuse in a command language environment. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI-92)*, pp. 628–632. Wiley, Chichester, 1992.

[8] G. S. Boolos and R. C. Jeffrey. *Computability and Logic*, 2nd edn. Cambridge University Press, Cambridge, 1980.

[9] F. M. Brown (ed.). *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*. Morgan Kaufmann, San Mateo, CA, 1987.

[10] G. Collins and L. Pryor. Achieving the functionality of filter conditions in a partial order planner. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92)*, pp. 375–380. AAAI/MIT Press, Cambridge, MA, 1992.

[11] E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, **18**, 453–457, 1975.

[12] D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems*, pp. 31–36. AAAI Press, Menlo Park, CA, 1994.

[13] G. W. Ernst and A. Newell. *GPS: A Case Study in Generality and Problem Solving*. Academic Press, New York, 1969.

[14] R. E. Fikes and N. J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, **2**, 189–208, 1971.

[15] R. W. Floyd. Assigning meaning to programs. In *Proceedings of the American Mathematics Society Symposia in Applied Mathematics*, **19**, 19–31, 1967.

[16] M. Gelfond, V. Lifschitz, and A. Rabinov. What are the limitations of the situation calculus? In *Automated Reasoning: Essays in Honor of Woddy Bledsoe*, R. Boyer, ed., pp. 167–179. Kluwer, Dordrecht, 1991.

[17] M. Gelfond and V. Lifschitz. Representing actions in extended logic programming. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pp. 559–573. MIT Press, Cambridge, MA, 1992.

[18] M. P. Georgeff. Actions, Processes, and Causality. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, M. P. Georgeff and A. L. Lansky, eds, pp. 99–122. Morgan Kaufmann, San Mateo, CA, 1987.

[19] C. Green. Application of theorem proving to problem solving. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence (IJCAI-69)*, pp. 219–239. Morgan Kaufmann, San Mateo, CA, 1969.

[20] G. Große. Propositional state event logic. In *Proceedings of the Joint European Workshop on Logics for Artificial Intelligence (JELIA '94)*, C. MaacNish, D. Peirce and L. M. Pereira, eds., pp. 316–331, Lecture Notes in Artificial Intelligence 838, Springer-Verlag, Berlin, 1994.

[21] A. R. Haas. The case for domain-specific frame axioms. In *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*, F. M. Brown, ed., pp. 343–348. Morgan Kaufmann, San Mateo, CA, 1987.

[22] S. Hanks and D. McDermott. Default reasoning, nonmonotonic logics, and the frame problem. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-86)*, pp. 328–333. AAAI/MIT Press, Cambridge, MA, 1986.

[23] S. Hanks and D. McDermott. Nonmonotonic logics and temporal projection. *Artificial Intelligence*, **33**, 379–412, 1987.

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

[25] P. Hayes. 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, San Francisco, CA, 1973.

[26] G. G. Hendrix. Modeling simultaneous actions and continuous processes. *Artificial Intelligence*, **4**, 145–180, 1975.

[27] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, **12**, 576–583, 1969.

[28] C. A. R. Hoare. Some properties of predicate transformers. *Journal of the ACM*, **25**, 461–480, 1978.

[29] G. N. Kartha. Soundness and completeness theorems for three formalizations of action. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, pp. 724–729. Morgan Kaufmann, San Mateo, CA, 1993.

[30] H. A. Kautz. *A First-Order Dynamic Logic for Planning*. Masters thesis, Department of Computer Science, University of Toronto, Toronto, Canada, 1982.

[31] H. A. Kautz. The logic of persistence. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-86)*, pp 401–405. AAAI/MIT Press, Cambridge, MA, 1986.

[32] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, in press.

[33] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic least-commitment planning. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pp. 1073–1078. AAAI Press, Menlo Park, CA, 1994.

[34] H. J. Levesque and R. J. Brachman. A fundamental tradeoff in knowledge representation and reasoning. In *Readings in Knowledge Representation*, H. J. Levesque and R. J. Brachman, eds, pp. 42–70. Morgan Kaufmann, San Mateo, CA, 1985.

[35] H. J. Levesque. Making believers out of computers. *Artificial Intelligence*, **30**, 81–108, 1986.

[36] V. Lifschitz. Pointwise circumscription: preliminary report. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-86)*, pp. 406–410. AAAI/MIT Press, Cambridge, MA, 1986.

[37] V. Lifschitz. Formal theories of action (preliminary report). In *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI-87)*, pp. 966–972. Morgan Kaufmann, San Mateo, CA, 1987.

[38] V. Lifschitz. Toward a metatheory of action. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pp. 376–386 Morgan Kaufmann, San Mateo, CA, 1991.

[39] V. Lifschitz. Restricted Monotonicity. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-93)*, pp. 432–437. AAAI/MIT Press, Cambridge, MA, 1993.

[40] F. Lin and Y. Shoham. Concurrent actions in the situation calculus. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92)*, pp. 590–595. AAAI/MIT Press, Cambridge, MA, 1992.

[41] L. Löwenheim. Über Möglichkeiten im Relativkalkül. *Mathematische Annalen*, **76**, 447–470, 1915.

[42] Z. Manna and R. Waldinger. *The Logical Basis for Computer Programming, Vol. 1: Deductive Reasoning*. Addison-Wesley, Reading, MA, 1985.

[43] Z. Manna and R. Waldinger. *The Logical Basis for Computer Programming, Vol. 2: Deductive Systems*. Addison-Wesley, Reading, MA, 1990.

[44] J. McCarthy. Programs with common sense. In *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, Her Majesty's Stationary Office, London, 1960.

[45] J. McCarthy 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, 1969.

[46] J. McCarthy. Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence*, **28**, 89–118, 1986.

[47] D. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, **6**, 101–155, 1982.

[48] D. McDermott. Reasoning about plans. In *Formal Theories of the Commonsense World*, J. R. Hobbs and R. C. Moore, eds, pp. 269–317. Ablex Publishing, Norwood, NJ, 1985.

[49] D. McDermott. Regression planning. *International Journal of Intelligent Systems*, **6**, 357–416, 1991.

[50] R. C. Moore. Reasoning about knowledge and action. Technical Note 191, Artificial Intelligence Center, SRI International, Menlo Park, CA, 1980.

[51] R. C. Moore. A formal theory of knowledge and action. In *Formal Theories of the Commonsense World*, J. R. Hobbs and R. C. Moore, eds, pp. 319–358. Ablex Publishing, Norwood, NJ, 1985.

[52] A. Newell, J. C. Shaw, and H. A. Simon. Report on a General Problem-Solving Program for a Computer. In *Proceedings of the International Conference on Information Processing*, pp. 256–264, UNESCO, Paris, 1960.

[53] E. P. D. Pednault. Preliminary report on a theory of plan synthesis. Technical Report 358, Artificial Intelligence Center, SRI International, Menlo Park, CA, 1985.

[54] E. P. D. Pednault. *Toward a Mathematical Theory of Plan Synthesis*. Ph.D. thesis, Department of Electrical Engineering, Stanford University, Stanford, CA, 1986.

[55] E. P. D. Pednault. Formulating multiagent, dynamic-world problems in the classical planning framework. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, M. P. Georgeff and A. L. Lansky, eds, pp. 47–82. Morgan Kaufmann, San Mateo, CA, 1987. Reprinted in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate, eds, pp. 675–710. Morgan Kaufmann, San Mateo, CA, 1990.

[56] E. P. D. Pednault. Extending conventional planning techniques to handle actions with context-dependent effects. In *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-88)*, pp. 55–59. AAAI/MIT Press, Cambridge, MA, 1988.

[57] E. P. D. Pednault. Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence*, **4**, 356–372, 1988.

[58] E. P. D. Pednault. ADL: exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, pp. 324–332. Morgan Kaufmann, San Mateo, CA, 1989.

[59] E. P. D. Pednault. Generalizing nonlinear planning to handle complex goals and actions with context-dependent effects. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pp. 240–254. Morgan Kaufmann, San Mateo, CA, 1991.

[60] J. S. Penberthy and D. S. Weld. UCPOP: a sound, complete, partial order planner for ADL. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pp. 103–114. Morgan Kaufmann, San Mateo, CA, 1992.

[61] J. S. Penberthy. *Planning with Continuous Change*. Ph.D. thesis, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 1993.

[62] J. S. Penberthy and D. S. Weld. Temporal Planning with Continuous Change. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pp. 1010–1015. AAAI/MIT Press, Cambridge, MA, 1994.

[63] V. R. Pratt. Semantical considerations on Floyd-Hoare logic. In *Proceedings of the 17th IEEE Symposium on Foundation of Computer Science*, pp. 109–121. IEEE Computer Society Press, Los Alamitos CA, 1976.

[64] L. Pryor and G. Collins. CASSANDRA: planning for contingencies. Technical Report 41, The Institute for the Learning Sciences, Northwestern University, Evanston, IL, 1993.

[65] R. Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, V. Lifschitz, ed., pp. 359–380. Academic Press, San Diego, CA, 1991.

[66] R. Reiter. The projection problem in the situation calculus: a soundness and completeness result, with an application to database updates. In *Proceedings of the 1st International Conference on Artificial Intelligence Planning Systems*, pp. 198–203. Morgan Kaufmann, San Mateo, CA, 1992.

[67] S. J. Rosenschein. Plan synthesis: a logical perspective. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI-81)*, pp. 331–337. Morgan Kaufmann, San Mateo, CA, 1981.

[68] R. B. Scherl and H. J. Levesque. The frame problem and knowledge-producing actions. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-93)*, pp. 689–695. AAAI/MIT Press, Cambridge, MA, 1993.

[69] L. K. Schubert. Monotonic solution of the frame problem in the situation calculus: an efficient method for worlds with fully specified actions. In *Knowledge Representation and Defeasible Reasoning*, H. E. Kyberg, R. P. Loui, and G. N. Carlson, eds, pp. 23–67. Kluwer Academic, Hingham, MA, 1990.

[70] J. R. Shoenfield. *Mathematical Logic*. Addison-Wesley, Reading, MA, 1967.

[71] Y. Shoham. Chronological ignorance: time, nonmonotonicity, necessity and causal theories. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-86)*, pp. 389–393. AAAI/MIT Press, Cambridge, MA, 1986.

[72] G. J. Sussman. A computational model of skill acquisition. Technical Report AI TR–197, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, 1973.

[73] J. D. Ullman. *Principles of Database Systems: 2nd edn.* Computer Science Press, Rockville, MD, 1982.

[74] D. Van Dalen. *Logic and Structure*. Springer-Verlag, Berlin, 1980.

[75] R. Waldinger. Achieving several goals simultaneously. In *Machine Intelligence 8*, E. Elcock and D. Michie, eds, pp. 94–136. Ellis Horwood, Edinburgh, Scotland, 1977.

[76] G. Wiederhold. *Database Design: Second Edition*. McGraw-Hill, New York, 1983.

Appendix

A Proofs of the Theorems

THEOREM A.1 (Theorem 5.5)
For every action a that can be described in ADL, (5.4)–(5.22) define a tidy regression operator for that action.

PROOF. It must be shown that $s \models a^{-1}(\varphi)$ if and only if $t \models \varphi$ for every pair of states $\langle s, t\rangle \in a$ and every formula φ in the same first-order language as a. This will be done by actually proving a stronger condition. Let θ^a be a function from states to states with the property that

1. $\mathrm{Dom}\big(\theta^a(s)\big) = \mathrm{Dom}(s)$.
2. $\big(\theta^a(s)\big)(\sigma) = s(\sigma)$ for every function and relation symbol σ.
3. $\big(\theta^a(s)\big)(x) = s(x)$ for every free variable x in the α, δ, and μ formulas that define the effects of action a.
4. $\big(\theta^a(s)\big)(x) \in \mathrm{Dom}(s)$ for all other variable symbols x, and for all such variables, s and t are two states such that $\mathrm{Dom}(s) = \mathrm{Dom}(t)$, then $\big(\theta^a(s)\big)(x) = \big(\theta^a(t)\big)(x)$.

Thus, the state $\theta^a(s)$ is the same as state s, except that different interpretations are possibly assigned to variables symbols other than those that appear as free variables in the α, δ, and μ formulas that define the effects of action a. Moreover, the interpretations are assigned as a function of the domain of s. Functions θ^a that satisfy the above conditions will be called *variable reassignments*.

To prove the theorem, we will first prove that $\theta^a(s) \models a^{-1}(\varphi)$ if and only if $\theta^a(t) \models \varphi$ for every action a that can be described in ADL, every formula φ, every pair of states $\langle s, t\rangle \in a$, and every variable reassignment θ^a. Because θ^a can be chosen such that $\theta^a(s) = s$, which implies $\theta^a t = t$, it follows that that $a^{-1}(\varphi)$ is a tidy regression operator

(i.e. $s \models a^{-1}(\varphi)$ if and only if $t \models \varphi$ for every formula φ and every pair of states $\langle s, t\rangle \in a$). The proof proceeds by induction on the number of connectives and quantifiers in φ. The base case consists of atomic formulas.

The proof for atomic formulas proceeds by induction on the number of function symbols that appear in the formula. Equations (5.4)–(5.10) cover the base cases, which consist of atomic formulas that do not contain any terms other than variable symbols for all relations other than equality. For equality relations, the base cases consists of all atomic equality formulas with at most one non-variable symbol in the term on the left-hand side of the equality symbol, and no terms other than variable symbols on the right-hand side.

For the atomic formulas TRUE and FALSE, $s \models \text{TRUE}$ if and only if $t \models \text{TRUE}$, and $s \models \text{FALSE}$ if and only if $t \models \text{FALSE}$ for any pair of states s and t. Therefore, $\theta^a(s) \models a^{-1}(\text{TRUE})$ if and only if $\theta^a(t) \models \text{TRUE}$, and $\theta^a(s) \models a^{-1}(\text{FALSE})$ if and only if $\theta^a(t) \models \text{FALSE}$, for every pair of states $\langle s, t\rangle \in a$ and every variable reassignment θ^a, where $a^{-1}(\text{TRUE})$ and $a^{-1}(\text{FALSE})$ are given by (5.4) and (5.5).

For atomic formulas of the form $R(z_1, \ldots, z_n)$, let $\langle s, t\rangle$ be any pair of states in a and θ^a be any variable reassignment. Then $\theta^a(t) \models R(z_1, \ldots, z_n)$ if and only if $\langle \theta^a(t)(z_1), \ldots, \theta^a(t)z_n)\rangle \in \theta^a(t)(R) = t(R)$. Moreover, because s and t have the same domains and variable interpretations, $\theta^a(s)$ and $\theta^a(t)$ assign the same interpretations to variable symbols. Therefore, $\theta^a(t) \models R(z_1, \ldots, z_n)$ if and only if $\langle \theta^a(s)(z_1), \ldots, \theta^a(s)(z_n)\rangle \in t(R)$. From (3.1) and (3.3),

$$t(R) = \Big\{\langle d_1, \ldots, d_n\rangle \in \text{Dom}(s)^n \mid s[d_1/x_1, \ldots, d_n/x_n] \models \alpha^a_R(x_1, \ldots, x_n) \vee \big(R(x_1, \ldots, x_n) \wedge \neg\delta^a_R(x_1, \ldots, x_n)\big)\Big\}.$$

Therefore, $\theta^a(t) \models R(z_1, \ldots, z_n)$ if and only if

$$s[\theta^a(s)(z_1)/x_1, \ldots, \theta^a(s)(z_1)/x_n] \models \alpha^a_R(x_1, \ldots, x_n) \vee \big(R(x_1, \ldots, x_n) \wedge \neg\delta^a_R(x_1, \ldots, x_n)\big).$$

Consider the states $\theta^a(s)[\theta^a(s)(z_1)/x_1, \ldots, \theta^a(s)(z_1)/x_n]$ and $s[\theta^a(s)(z_1)/x_1, \ldots, \theta^a(s)(z_1)/x_n]$. Both assign the same interpretations to the function and relation symbols, and to the variables $x_1, \ldots, x_n$. By the definition of a variable reassignment, they also assign the same interpretations to all other free variables in $\alpha^a_R(x_1, \ldots, x_n)$ and $\delta^a_R(x_1, \ldots, x_n)$. The two states therefore assign the same interpretations to all symbols that affect the truth value of $\alpha^a_R(x_1, \ldots, x_n) \vee \big(R(x_1, \ldots, x_n) \wedge \neg\delta^a_R(x_1, \ldots, x_n)\big)$. Hence, the truth value of this formula must be the same in both states. Therefore, $\theta^a(t) \models R(z_1, \ldots, z_n)$ if and only if

$$\theta^a(s)[\theta^a(s)(z_1)/x_1, \ldots, \theta^a(s)(z_1)/x_n] \models \alpha^a_R(x_1, \ldots, x_n) \vee \big(R(x_1, \ldots, x_n) \wedge \neg\delta^a_R(x_1, \ldots, x_n)\big).$$

Because safe substitution has the property that $s[s(\tau)/x] \models \varphi$ if and only if $s \models [\tau/x]\varphi$ for any term τ and any variable x, it follows that $\theta^a(t) \models R(z_1, \ldots, z_n)$ if and only if

$$\theta^a(s) \models \alpha^a_R(z_1, \ldots, z_n) \vee \big(R(z_1, \ldots, z_n) \wedge \neg\delta^a_R(z_1, \ldots, z_n)\big).$$

Therefore, for every n-ary relation symbol R, every pair of states $\langle s, t\rangle \in a$, and every variable reassignment θ^a, $\theta^a(s) \models a^{-1}\big(R(z_1, \ldots, z_n)\big)$ if and only if $\theta^a(t) \models R(z_1, \ldots, z_n)$, where $a^{-1}\big(R(z_1, \ldots, z_n)\big)$ is given by (5.6). Equation (5.8) is a special case of (5.6); therefore, for every zero-place relation symbol (i.e. proposition symbol) P, every pair of states $\langle s, t\rangle \in a$, and every variable reassignment θ^a, $\theta^a(s) \models a^{-1}\big(P\big)$ if and only if $\theta^a(t) \models P$, where $a^{-1}\big(P\big)$ is given by (5.8).

Through a virtually identical line of reasoning, one can show on the basis of (3.2) and (3.4) that, for every n-ary function symbol F, every pair of states $\langle s, t\rangle \in a$, and every variable reassignment θ^a, $\theta^a(s) \models a^{-1}\big(F(z_1, \ldots, z_n) = w\big)$ if and only if $\theta^a(t) \models \big(F(z_1, \ldots, z_n) = w\big)$, where $a^{-1}\big(F(z_1, \ldots, z_n) = w\big)$ is given by (5.7). Equation (5.9) is a special case of (5.7); therefore, for every zero-place function symbol (i.e. 'constant' symbol) C, every pair of states $\langle s, t\rangle \in a$, and every variable reassignment θ^a, $\theta^a(s) \models a^{-1}\big(C = w\big)$ if and only if $\theta^a(t) \models \big(C = w\big)$, where $a^{-1}\big(C = w\big)$ is given by (5.8).

The final base case consists of atomic formulas of the form $z_1 = z_2$. From the semantics presented in Section 2, $\theta^a(s) \models \left(z_1 = z_2\right)$ if and only if $\theta^a(s)(z_1) = \theta^a(s)(z_2)$. Since $\theta^a(s)$ and $\theta^a(t)$ assign the same interpretations to variables, $\theta^a(t)(z_1) = \theta^a(s)(z_1)$ and $\theta^a(t)(z_2) = \theta^a(s)(z_2)$. Hence, $\theta^a(s) \models \left(z_1 = z_2\right)$ if and only if $\theta^a(t)(z_1) = \theta^a(t)(z_2)$, which is true if and only if $\theta^a(t) \models \left(z_1 = z_2\right)$. Therefore, for every pair of states $\langle s, t\rangle \in a$ and every variable reassignment θ^a, $\theta^a(s) \models a^{-1}\left(z_1 = z_2\right)$ if and only if $\theta^a(t) \models \left(z_1 = z_2\right)$, where $a^{-1}\left(z_1 = z_2\right)$ is given by (5.10).

Equations (5.11)–(5.15) cover the induction steps for the atomic formulas. The proof makes use of the following lemma: if $\lambda(z)$ is a formula containing the free variable z and if z does not appear in the term τ, then for every state s, $s \models \lambda(\tau)$ if and only if $s \models \exists z(\tau = z \wedge \lambda(z))$. Proof: from the properties of safe substitution, $s \models \lambda(\tau)$ if and only if $s[s(\tau)/z] \models \lambda(z)$. The latter is true if and only if there exists an element $d \in \mathrm{Dom}(s)$ such that $s(\tau) = d$ and $s[d/z] \models \lambda(z)$. In turn, this is true if and only if there exists an element $d \in \mathrm{Dom}(s)$ such that $s[d/z](\tau) = s[d/z](z)$ and $s[d/z] \models \lambda(z)$, which is true if and only if $s \models \exists z(\tau = z \wedge \lambda(z))$. Hence, for every state s, $s \models \lambda(\tau)$ if and only if $s \models \exists z(\tau = z \wedge \lambda(z))$.

For the induction step, let us first consider atomic formulas of the forms $R(\ldots)$ and $F(\ldots) = w$, where w is a variable symbol. Assume that $\theta^a(s) \models a^{-1}(\varphi)$ if and only if $\theta^a(t) \models \varphi$ for every pair of states $\langle s, t\rangle \in a$, every variable reassignment θ^a, and every atomic formula φ of the form $R(\ldots)$ or $F(\ldots) = w$, where w is a variable symbol and where $R(\ldots)$ and $F(\ldots)$ contain no more than k function symbols in their argument lists. Let $R(\ldots, \tau, \ldots)$ be an atomic formula containing $k + 1$ function symbols, and let τ be a term containing one or more of these symbols. Let z be a variable symbol that does not appear in $R(\ldots, \tau, \ldots)$ nor as a free variable in any of the α, δ, or μ formulas that define the effects of action a. Let $\langle s, t\rangle$ be any pair of states in a and θ^a be any variable reassignment. By the lemma presented above, $\theta^a(t) \models R(\ldots, \tau, \ldots)$ if and only if for some $d \in \mathrm{Dom}\left(\theta^a(t)\right)$, $\theta^a(t)[d/z] \models \left(\tau = z\right)$ and $\theta^a(t)[d/z] \models R(\ldots, z, \ldots)$. Because z is not a free variable in any of the α, δ, or μ formulas that define the effects of action a, there exists a variable reassignment $\hat{\theta}^a$ such that $\hat{\theta}^a(t) = \theta^a(t)[d/z]$. Let $\hat{\theta}^a$ be this variable reassignment. Because τ by definition must be of the form $F(\ldots)$, where $F(\ldots)$ contains no more than k function symbols in its argument list (this includes functions F of zero arguments), it follows from the induction hypothesis that $\hat{\theta}^a(t)[d/z] \models \left(\tau = z\right)$ if and only if $\hat{\theta}^a(s)[d/z] \models a^{-1}\left(\tau = z\right)$. It likewise follows from the induction hypothesis that $\hat{\theta}^a(t)[d/z] \models R(\ldots, z, \ldots)$ if and only if $\hat{\theta}^a(s)[d/z] \models a^{-1}\left(R(\ldots, z, \ldots)\right)$. Therefore, $\theta^a(t) \models R(\ldots, \tau, \ldots)$ if and only if $\hat{\theta}^a(s)[d/z] \models a^{-1}\left(\tau = z\right)$ and $\hat{\theta}^a(s)[d/z] \models a^{-1}\left(R(\ldots, z, \ldots)\right)$ for some $d \in \mathrm{Dom}\left(\theta^a(t)\right)$. Since states s and t have the same domains and variable interpretations, and since $\hat{\theta}^a(t) = \theta^a(t)[d/z]$, it follows that $\hat{\theta}^a(s) = \theta^a(s)[d/z]$. In addition, $\mathrm{Dom}\left(\theta^a(s)\right) = \mathrm{Dom}\left(\theta^a(t)\right)$. Therefore, $\theta^a(t) \models R(\ldots, \tau, \ldots)$ if and only if, for some $d \in \mathrm{Dom}\left(\theta^a(s)\right)$, $\theta^a(s)[d/z] \models a^{-1}\left(\tau = z\right)$ and $\theta^a(s)[d/z] \models a^{-1}\left(R(\ldots, z, \ldots)\right)$. The latter is true if and only if $\theta^a(s) \models \exists z\left[a^{-1}\left(\tau = z\right) \wedge a^{-1}\left(R(\ldots, z, \ldots)\right)\right]$. Hence, $\theta^a(s) \models a^{-1}\left(R(\ldots, \tau, \ldots)\right)$ if and only if $\theta^a(t) \models R(\ldots, \tau, \ldots)$ for every pair of states $\langle s, t\rangle \in a$ and every variable reassignment θ^a, where $R(\ldots, \tau, \ldots)$ is an atomic formula containing $k + 1$ function symbols, τ is a term that is not a variable symbol, and $a^{-1}\left(R(\ldots, \tau, \ldots)\right)$ is given by (5.11). Therefore, by induction on k, $\theta^a(s) \models a^{-1}\left(R(\ldots)\right)$ if and only if $\theta^a(t) \models R(\ldots)$ for every pair of states $\langle s, t\rangle \in a$, every variable reassignment θ^a, and every atomic formula of the form $R(\ldots)$, where $a^{-1}\left(R(\ldots)\right)$ is given by (5.6), (5.8) or (5.11), depending on the terms in the argument list of $R(\ldots)$.

By a virtually identical line of reasoning, it can be shown that $\theta^a(s) \models a^{-1}\left(F(\ldots) = w\right)$ if and only if $\theta^a(t) \models \left(F(\ldots) = w\right)$ for every pair of states $\langle s, t\rangle \in a$, every variable reassignment θ^a, and every atomic formula of the form $F(\ldots) = w$, where w is a variable symbol and $a^{-1}\left(F(\ldots) = w\right)$ is given by (5.7), (5.9) or (5.12), depending on the terms in the argument list of $F(\ldots)$.

Now consider atomic formulas of the form $F(\ldots) = \tau$, where τ is a term that is not a variable symbol. Let z be a variable symbol that does not appear in $F(\ldots) = \tau$. Let $\langle s, t\rangle$ be any pair of states in a and θ^a be any variable reassignment. By the lemma presented above, $\theta^a(t) \models \left(F(\ldots) = \tau\right)$ if and only if, for some $d \in \mathrm{Dom}\left(\theta^a(t)\right)$, $\theta^a(t)[d/z] \models \left(\tau = z\right)$ and $\theta^a(t)[d/z] \models \left(F(\ldots) = z\right)$. Since τ is not a variable symbol, the formula $\tau = z$ must be of the form $G(\ldots) = z$ for some function symbol G (possibly a zero-place function). Therefore, from the preceding proof steps, $\theta^a(t)[d/z] \models \left(\tau = z\right)$ if and only if $\theta^a(s)[d/z] \models a^{-1}\left(\tau = z\right)$. Likewise, $\theta^a(t)[d/z] \models \left(F(\ldots) = z\right)$ if and only if $\theta^a(s)[d/z] \models a^{-1}\left(F(\ldots) = z\right)$. In addition, since s and t have the same domains, $\mathrm{Dom}\left(\theta^a(t)\right) = \mathrm{Dom}\left(\theta^a(s)\right)$. Therefore, $\theta^a(t) \models \left(F(\ldots) = \tau\right)$ if and only if, for some $d \in \mathrm{Dom}\left(\theta^a(s)\right)$, $\theta^a(s)[d/z] \models a^{-1}\left(\tau = z\right)$ and $\theta^a(s)[d/z] \models a^{-1}\left(F(\ldots) = z\right)$, which is true if and only

only if $\theta^a(s) \models \exists z\left[a^{-1}\left(\tau = z\right) \wedge a^{-1}\left(F(\ldots) = z\right)\right]$. Hence, $\theta^a(s) \models a^{-1}\left(F(\ldots) = \tau\right)$ if and only if $\theta^a(t) \models \left(F(\ldots) = \tau\right)$ for every pair of states $\langle s, t\rangle \in a$, every variable reassignment θ^a, and every atomic formula of the form $F(\ldots) = \tau$, where τ is a term that is not a variable symbol and $a^{-1}\left(F(\ldots) = \tau\right)$ is given by (5.13). Equation 5.14 is a special case of (5.13); therefore, $\theta^a(s) \models a^{-1}\left(C = \tau\right)$ if and only if $\theta^a(t) \models \left(C = \tau\right)$ for every pair of states $\langle s, t\rangle \in a$, every variable reassignment θ^a, where C is a zero-place function symbol, τ is a term that is not a variable symbol, and $a^{-1}\left(C = \tau\right)$ is given by (5.14).

All remaining atomic formulas are of the form $w = \tau$, where w is a variable symbol and τ is a term other than a variable symbol. Let $\langle s, t\rangle$ be any pair of states in a and θ^a be any variable reassignment. Then $\theta^a(t) \models \left(w = \tau\right)$ if and only if $\theta^a(t) \models \left(\tau = w\right)$. Since τ is not a variable symbol, it must be of the form $F(\ldots)$. Moreover, it has already been shown that $\theta^a(t) \models \left(F(\ldots) = w\right)$ if and only if $\theta^a(s) \models a^{-1}\left(F(\ldots) = w\right)$. Therefore, $\theta^a(s) \models a^{-1}\left(w = \tau\right)$ if and only if $\theta^a(t) \models \left(w = \tau\right)$ for every pair of states $\langle s, t\rangle \in a$, every variable reassignment θ^a, and every atomic formula of the form $w = \tau$, where τ is a term that is not a variable symbol and $a^{-1}\left(w = \tau\right)$ is given by (5.15).

We have thus far shown that $\theta^a(s) \models a^{-1}(\varphi)$ if and only if $\theta^a(t) \models \varphi$ for every pair of states $\langle s, t\rangle \in a$, every variable reassignment θ^a, and every atomic formula φ. The proof for all remaining formulas proceeds by induction on the number of connectives and quantifiers in φ. Assume that the above condition holds for all formulas φ containing no more than k connectives and quantifiers. Let λ be a formula containing $k + 1$ connectives and quantifiers. Let $\langle s, t\rangle$ be any pair of states in a and θ^a be any variable reassignment.

Consider the case in which λ is of the form $\neg\varphi$, where φ is a formula that contains k connectives and quantifiers. From the semantics presented in Section 2, $\theta^a(t) \models \neg\varphi$ if and only if $\theta^a(t) \not\models \varphi$. By the induction hypothesis, $\theta^a(t) \models \varphi$ if and only if $\theta^a(s) \models a^{-1}(\varphi)$. Therefore, $\theta^a(t) \models \neg\varphi$ if and only if $\theta^a(s) \not\models a^{-1}(\varphi)$, which is true if and only if $\theta^a(s) \models \neg a^{-1}(\varphi)$. Hence, $\theta^a(s) \models a^{-1}(\neg\varphi)$ if and only if $\theta^a(t) \models \neg\varphi$, where $a^{-1}(\neg\varphi)$ is given by Equation 5.16.

Next, consider the case in which λ is of the form $\varphi \wedge \psi$, where the formulas φ and ψ each contain no more than k connectives and quantifiers, and between them they have a total of exactly k connectives and quantifiers. From Section 2, $\theta^a(t) \models \varphi \wedge \psi$ if and only if $\theta^a(t) \models \varphi$ and $\theta^a(t) \models \psi$. By the induction hypothesis, $\theta^a(t) \models \varphi$ if and only if $\theta^a(s) \models a^{-1}(\varphi)$ and $\theta^a(t) \models \psi$ if and only if $\theta^a(s) \models a^{-1}(\psi)$. Hence, $\theta^a(t) \models \varphi \wedge \psi$ if and only if $\theta^a(s) \models a^{-1}(\varphi)$ and $\theta^a(s) \models a^{-1}(\psi)$, which is true if and only if $\theta^a(s) \models a^{-1}(\varphi) \wedge a^{-1}(\psi)$. Therefore, $\theta^a(s) \models a^{-1}(\varphi \wedge \psi)$ if and only if $\theta^a(t) \models \varphi \wedge \psi$, where $a^{-1}(\varphi \wedge \psi)$ is given by Equation 5.17.

By similar lines of reasoning for the other connectives: $\theta^a(s) \models a^{-1}(\varphi \vee \psi)$ if and only if $\theta^a(t) \models \varphi \vee \psi$, where $a^{-1}(\varphi \vee \psi)$ is given by (5.18); $\theta^a(s) \models a^{-1}(\varphi \rightarrow \psi)$ if and only if $\theta^a(t) \models \varphi \rightarrow \psi$, where $a^{-1}(\varphi \rightarrow \psi)$ is given by (5.19); and $\theta^a(s) \models a^{-1}(\varphi \leftrightarrow \psi)$ if and only if $\theta^a(t) \models \varphi \leftrightarrow \psi$, where $a^{-1}(\varphi \leftrightarrow \psi)$ is given by (5.20).

Next, consider the case in which λ is of the form $\forall x\,\varphi$, where the formula φ contains k connectives and quantifiers. Let z be a variable symbol (possibly x itself) that does not appear as a free variable in $\forall x\,\varphi$ nor in any of the α, δ, or μ formulas that define the effects of action a. We will first show that $\theta^a(t) \models \forall x\,\varphi$ if and only if $\theta^a(t) \models \forall z\,[z/x]\varphi$. If z is in fact variable x, then the result follows immediately. Suppose instead that z and x are distinct variables. Then z does not appear as a free variable in φ. From the semantics presented in Section 2, $\theta^a(t) \models \forall x\,\varphi$ if and only if $\theta^a(t)[d/x] \models \varphi$ for every element $d \in \mathrm{Dom}\left(\theta^a(t)\right)$. Because z is not a free variable in φ, we can assign the value d to z without affecting the truth value of φ. Therefore, $\theta^a(t)[d/x] \models \varphi$ if and only if $\left(\theta^a(t)[d/z]\right)\left[\left(\theta^a(t)[d/z]\right)(z)/x\right] \models \varphi$. From the properties of safe substitution, the latter is true if and only if $\left(\theta^a(t)[d/z]\right) \models [z/x]\varphi$. Therefore, $\theta^a(t) \models \forall x\,\varphi$ if and only if, for every element $d \in \mathrm{Dom}\left(\theta^a(t)\right)$, $\theta^a(t)[d/z] \models [z/x]\varphi$. The latter is true if and only if $\theta^a(t) \models \forall z\,[z/x]\varphi$. Thus, independent of whether z is identical or different from x, $\theta^a(t) \models \forall x\,\varphi$ if and only if $\theta^a(t)[d/z] \models [z/x]\varphi$ for every element $d \in \mathrm{Dom}\left(\theta^a(t)\right)$.

Because z is not a free variable in any of the α, δ, or μ formulas that define the effects of action a, there exists a variable reassignment $\hat{\theta}^a$ such that $\hat{\theta}^a(t) = \theta^a(t)[d/z]$. Let $\hat{\theta}^a$ be this variable reassignment. By the induction hypothesis, it therefore follows that $\hat{\theta}^a(t) \models [z/x]\varphi$ if and only if $\hat{\theta}^a(s) \models a^{-1}\left([z/x]\varphi\right)$. Since states s and t have the same domains and variable interpretations, and since $\hat{\theta}^a(t) = \theta^a(t)[d/z]$, it follows that $\hat{\theta}^a(s) = \theta^a(s)[d/z]$. In addition, $\mathrm{Dom}\left(\theta^a(s)\right) = \mathrm{Dom}\left(\theta^a(t)\right)$. Therefore, $\theta^a(t)[d/z] \models [z/x]\varphi$ if and only if $\theta^a(s)[d/z] \models a^{-1}\left([z/x]\varphi\right)$. From this we may conclude that $\theta^a(t) \models \forall x\,\varphi$ if and only if $\theta^a(s)[d/z] \models a^{-1}\left([z/x]\varphi\right)$ for every element $d \in \mathrm{Dom}\left(\theta^a(s)\right)$, which is true if and only if $\theta^a(s) \models \forall z\,a^{-1}\left([z/x]\varphi\right)$. Therefore, $\theta^a(s) \models a^{-1}\left(\forall x\,\varphi\right)$ if and only if $\theta^a(t) \models \forall x\,\varphi$, where $a^{-1}\left(\forall x\,\varphi\right)$ is given by (5.21).

By a similar line of reasoning, we can likewise show that $\theta^a(s) \models a^{-1}\left(\exists x\, \varphi\right)$ if and only if $\theta^a(t) \models \exists x\, \varphi$, where the formula φ contains k connectives and quantifiers and where $a^{-1}\left(\exists x\, \varphi\right)$ is given by (5.22).

We have shown that if $\theta^a(s) \models a^{-1}(\varphi)$ if and only if $\theta^a(t) \models \varphi$ for every pair of states $\langle s, t\rangle \in a$, every variable reassignment θ^a, and every formula φ containing no more than k connectives and quantifiers, then the same holds for every formula containing $k+1$ connectives and quantifiers. Therefore, by induction, $\theta^a(s) \models a^{-1}(\varphi)$ if and only if $\theta^a(t) \models \varphi$ for every pair of states $\langle s, t\rangle \in a$, every variable reassignment θ^a, and every formula φ. For each pair of states $\langle s, t\rangle \in a$, let us now choose θ^a such that $\theta^a(s) = s$. Because s and t have the same domains and variable interpretations, it then follows that $\theta^a(t) = t$. Therefore $s \models a^{-1}(\varphi)$ if and only if $t \models \varphi$ for every pair of states $\langle s, t\rangle \in a$ and every formula φ. ∎

To show that (5.1)–(5.3) do in fact provide an equivalent axiomatization for actions described in ADL, consider again the augmented language L' defined in Section 5. The states that define the semantic structures for this language consist of a domain of objects, an interpretation assigned to each variable symbol, a pair of interpretations assigned to each pair of primed and non-primed relation symbols R' and R, and a pair of interpretations assigned to each primed–non-primed pair of function symbols F' and F. Each state u' of L' can therefore be decomposed in to a pair of states s and t of the original language L such that

1. The domains of s and t are identical and equal to the domain of u' (i.e. $\mathrm{Dom}(s) = \mathrm{Dom}(t) = \mathrm{Dom}(u')$).
2. The interpretations assigned to the variable symbols by s and t are identical and equal to the interpretations assigned by u' (i.e. $s(x) = t(x) = u'(x)$ for every variable symbol x).
3. For every pair of primed and non-primed relation symbols R' and R, the interpretation assigned to R by s is the interpretation assigned to R by u', and the interpretation assigned to R by t is the interpretation assigned to R' by u' (i.e. $s(R) = u'(R)$ and $t(R) = u'(R')$).
4. For every pair of primed and non-primed function symbols F' and F, the interpretation assigned to F by s is the interpretation assigned to F by u', and the interpretation assigned to F by t is the interpretation assigned to F' by u' (i.e. $s(F) = u'(F)$ and $t(F) = u'(F')$).

Likewise, a pair of states s and t of language L with the same domain and interpretations of the variable symbols can be combined into a state of language L', denoted $s\|t$, such that

1. $\mathrm{Dom}(s\|t) = \mathrm{Dom}(s) = \mathrm{Dom}(t)$.
2. $s\|t(x) = s(x) = t(x)$ for every variable symbol x.
3. $s\|t(R) = s(R)$ and $s\|t(R') = t(R)$ for every pair of primed and non-primed relation symbols R' and R.
4. $s\|t(F) = s(F)$ and $s\|t(F') = t(F)$ for every pair of primed and non-primed function symbols F' and F.

Note that the composite state $s\|t$ exists if and only if s and t have the same domain and interpretations of the variable symbols. This notion of composite states provides the basis for demonstrating that (5.1)–(5.3) provide an equivalent axiomatization for actions described in ADL.

THEOREM A.2 (Theorem 5.6)
For any action a that can be described in ADL and any pair of states $s, t \in \mathcal{W}$, it is the case that $\langle s, t\rangle \in a$ if and only if $s\|t$ exists and $s\|t \models \Omega^a$, where Ω^a consists of (5.1)–(5.3).

PROOF. Suppose that $\langle s, t\rangle \in a$. Then $s\|t$ must exist, since the semantics of ADL require that s and t have the same domains and interpretations of the variable symbols. The semantics of ADL also require that $s \models \pi^a$. By construction, this implies that $s\|t \models \pi^a$. Hence, (5.1) is satisfied. Now consider (5.2). From the semantic definitions introduced in Section 2,

$$s\|t \models \forall x_1, \ldots, x_n \left[R'(x_1, \ldots, x_n) \leftrightarrow \varphi_R^a(x_1, \ldots, x_n)\right]$$

if and only if

$$\text{for all } d_1, \ldots, d_n \in \mathrm{Dom}(s\|t),\ s\|t[d_1/x_1, \ldots, d_n/x_n] \models R'(x_1, \ldots, x_n)$$
$$\text{precisely when } s\|t[d_1/x_1, \ldots, d_n/x_n] \models \varphi_R^a(x_1, \ldots, x_n)$$

if and only if

$$\text{for all } d_1, \ldots, d_n \in \mathrm{Dom}(s\|t),\ \langle d_1, \ldots, d_n\rangle \in s\|t(R')$$
$$\text{precisely when } s\|t[d_1/x_1, \ldots, d_n/x_n] \models \varphi_R^a(x_1, \ldots, x_n).$$

From the definition of $s\|t$, $s\|t(R') = t(R)$. Also, because $\varphi_R^a(x_1, \ldots, x_n)$ does not contain any primed function or relation symbols, its truth value in state $s\|t$ does not depend on the interpretations assigned to the primed symbols by t. Consequently, $s\|t[d_1/x_1, \ldots, d_n/x_n] \models \varphi_R^a(x_1, \ldots, x_n)$ if and only if $s[d_1/x_1, \ldots, d_n/x_n] \models \varphi_R^a(x_1, \ldots, x_n)$.

From these simplifications, and the fact that s, t, and $s\|t$ all have the same domain,

$$s\|t \models \forall x_1, \ldots, x_n \left[R'(x_1, \ldots, x_n) \leftrightarrow \varphi_R^a(x_1, \ldots, x_n)\right]$$

if and only if

$$\text{for all } d_1, \ldots, d_n \in \text{Dom}(s\|t),\ \langle d_1, \ldots, d_n\rangle \in t(R)$$
$$\text{precisely when } s[d_1/x_1, \ldots, d_n/x_n] \models \varphi_R^a(x_1, \ldots, x_n)$$

if and only if

$$t(R) = \left\{\langle d_1, \ldots, d_n\rangle \in \text{Dom}(s)^n \mid s[d_1/x_1, \ldots, d_n/x_n] \models \varphi_R^a(x_1, \ldots, x_n)\right\}.$$

Therefore, $s\|t$ satisfies (5.2) for all relation symbols R if and only if the interpretation of R in state t satisfies the semantics of ADL as given by (3.1). Since $\langle s, t\rangle \in a$, (3.1) must be satisfied and, hence, $s\|t$ must satisfy (5.2) for all relation symbols R. By a similar line of reasoning, it likewise follows that

$$s\|t \models \forall x_1, \ldots, x_n, y \left[F'(x_1, \ldots, x_n) = y \leftrightarrow \varphi_F^a(x_1, \ldots, x_n, y)\right]$$

if and only if

$$t(F) = \left\{\langle d_1, \ldots, d_n, e\rangle \in \text{Dom}(s)^{n+1} \mid s[d_1/x_1, \ldots, d_n/x_n, e/y] \models \varphi_F^a(x_1, \ldots, x_n, y)\right\}.$$

Therefore, $s\|t$ satisfies (5.3) for all function symbols F if and only if the interpretation of F in state t satisfies the semantics of ADL as given by (3.2). Since $\langle s, t\rangle \in a$, (3.2) must be satisfied and, hence, $s\|t$ must satisfy (5.3) for all function symbols F. Thus, if $\langle s, t\rangle \in a$, then $s\|t$ exists and $s\|t \models \Omega^a$.

Now suppose that $s\|t$ exists and $s\|t \models \Omega^a$. Then s and t must have the same domains and interpretations of the variable symbols. By (5.1), it must be the case that $s\|t \models \pi^a$. Hence, $s \models \pi^a$ by the definition of $s\|t$. From the discussion above, it must also be the case that (3.1) and (3.2) are satisfied for all relation symbols R and all function symbols F, respectively. The pair of states $\langle s, t\rangle$ therefore satisfies the semantics of the ADL description of action a. Hence, $\langle s, t\rangle \in a$. ∎

THEOREM A.3 (Theorem 6.1)
A regression operator a^{-1} for an action a is tidy if and only if, for every formula φ, $a^{-1}(\neg\varphi)$ and $\neg a^{-1}(\varphi)$ have the same truth value in every state in which action a can be executed (i.e. for every formula φ and every pair of states $\langle s, t\rangle \in a$, $s \models a^{-1}(\neg\varphi)$ if and only if $s \models \neg a^{-1}(\varphi)$).

PROOF. Suppose that a^{-1} is a tidy regression operator for action a. Let φ be any formula and $\langle s, t\rangle \in a$ be any pair of states in action a. Then $s \models a^{-1}(\varphi)$ if and only if $t \models \varphi$. Therefore, $s \not\models a^{-1}(\varphi)$ if and only if $t \not\models \varphi$, and hence $s \models \neg a^{-1}(\varphi)$ if and only if $t \models \neg\varphi$. But, since a^{-1} is a tidy, $s \models a^{-1}(\neg\varphi)$ if and only if $t \models \neg\varphi$. Therefore, $s \models a^{-1}(\neg\varphi)$ if and only if $s \models \neg a^{-1}(\varphi)$. Moreover, this holds for every formula φ and every pair of states $\langle s, t\rangle \in a$, since their choices were arbitrary. Hence, if a^{-1} is a tidy regression operator for action a, then for every formula φ, $a^{-1}(\neg\varphi)$ and $\neg a^{-1}(\varphi)$ have the same truth value in every state in which action a can be executed.

To prove the converse, suppose that $s \models a^{-1}(\neg\varphi)$ if and only if $s \models \neg a^{-1}(\varphi)$ for every formula φ and every pair of states $\langle s, t\rangle \in a$. Let φ be any formula and $\langle s, t\rangle$ be any pair of states in action a. By the definition of a regression operator, if $s \models a^{-1}(\neg\varphi)$, then $t \models \neg\varphi$. Since $s \models a^{-1}(\neg\varphi)$ if and only if $s \models \neg a^{-1}(\varphi)$, this implies that if $s \models \neg a^{-1}(\varphi)$, then $t \models \neg\varphi$. This in turn implies that if $s \not\models a^{-1}(\varphi)$, then $t \not\models \varphi$. By the definition of a regression operator, it is also the case that if $s \models a^{-1}(\varphi)$, then $t \models \varphi$. Therefore, $s \models a^{-1}(\varphi)$ if and only if $t \models \varphi$. Moreover, this holds for every formula φ and every pair of states $\langle s, t\rangle \in a$. Therefore, if for every formula φ, $a^{-1}(\neg\varphi)$ and $\neg a^{-1}(\varphi)$ have the same truth value in every state in which action a can be executed, then a^{-1} is a tidy regression operator. Hence, the converse is true. ∎

THEOREM A.4 (Theorem 6.4)
If an action has a tidy regression operator, then that action is quasi-deterministic.

PROOF. Suppose that action a has a tidy regression operator a^{-1}. Assume that a is not quasi-deterministic. Then there must exist a pair of states $\langle s, t\rangle \in a$ and a pair of states $\langle s', t'\rangle \in a$ such that s and s' are elementary equivalent, but t and t' are not. Let $\langle s, t\rangle$ and $\langle s', t'\rangle$ be two such pairs of states. Since t and t' are not elementary equivalent, there must exist a formula φ such that $t \models \varphi$ and $t' \not\models \varphi$. Let φ be such a formula. Since a^{-1} is a tidy regression operator,

$s \models a^{-1}(\varphi)$ if and only if $t \models \varphi$, and $s' \models a^{-1}(\varphi)$ if and only if $t' \models \varphi$. But $t \models \varphi$ and $t' \not\models \varphi$. Hence, $s \models a^{-1}(\varphi)$ and $s' \not\models a^{-1}(\varphi)$. This, however, contradicts the fact that s and s' are elementary equivalent, since the formula given by $a^{-1}(\varphi)$ must either be true in both s and s' or false in both s and s'. Therefore, the assumption that a is not quasi-deterministic is false. Hence, if action a has a tidy regression operator, then a must be quasi-deterministic. ∎

THEOREM A.5 (Theorem 6.6)
If a is an action described in ADL and a^{-1} is the regression operator defined by (5.4)–(5.22), then a^{-1} is a coherent regression operator for action a.

PROOF. Let a^{-1} be a regression operator defined by (5.4)–(5.22). From (5.17), $a^{-1}(\varphi \wedge \psi) \equiv a^{-1}(\varphi) \wedge a^{-1}(\psi)$. Hence, $a^{-1}(\varphi \wedge \psi) \models a^{-1}(\varphi) \wedge a^{-1}(\psi)$. Therefore, a^{-1} satisfies Clause 2 of Definition 6.5. To show that Clause 1 also holds, let φ and ψ be an arbitrary pair of formulas such that $\varphi \models \psi$. Let a' be the action obtained from a by replacing the precondition of execution π^a of a with the formula TRUE. Then a^{-1} is also the regression operator for a', as defined by (5.4)–(5.22). Let $\langle s, t\rangle$ be an arbitrary pair of states in a'. Then $s \models a^{-1}(\varphi)$ if and only if $t \models \varphi$, and $s \models a^{-1}(\psi)$ if and only if $t \models \psi$. But, since $\varphi \models \psi$, if $t \models \varphi$, then $t \models \psi$. Hence, if $s \models a^{-1}(\varphi)$, then $s \models a^{-1}(\psi)$. Moreover, this must hold for every pair of states $\langle s, t\rangle \in a'$. By the semantics of ADL,

$$\{s \mid \exists t\, \langle s, t\rangle \in a'\} = \{s \mid s \models \pi^{a'}\} = \{s \mid s \models \mathrm{TRUE}\} = \mathcal{W}.$$

Hence, for every state s, if $s \models a^{-1}(\varphi)$, then $s \models a^{-1}(\psi)$. Therefore, $a^{-1}(\varphi) \models a^{-1}(\psi)$, thus satisfying Clause 1 of Definition 6.5. The regression operator a^{-1} is therefore coherent. ∎

The proof of Theorem 7.3 is based on work by Ackermann [1] and Löwenheim [41] regarding the decidability of *equality formulas*. Equality formulas are characterized by their atomic subformulas, which are of the form $x_1 = x_2$, $x_1 = C_1$, $C_1 = x_1$, or $C_1 = C_2$, where x_1 and x_2 are variable symbols, and C_1 and C_2 are constant symbols (for the sake of brevity, the term 'constant symbol' will be used in this section instead of 'zero-place function symbol'; however, the reader should keep in mind that the values of these symbols can change as the result of performing an action). By extending the work of Löwenheim, Ackermann proved a theorem which implies that if an equality formula is true in a state whose domain is infinite, then that formula is also true in a state whose domain is finite. By making use of this fact, it is possible to construct an action representable in ADL that does not have a tidy progression operator. Ackermann's proof, though, relies upon the standard semantics of first-order logic, wherein free variables are treated as though they are universally quantified. To make use of Ackermann's results, they must therefore be restated and proved for the semantics given in Section 2.

Ackermann's results are stated and proved in terms of equality formulas that are in *prenex normal form* [70, 8, 74, 42, 43]. Such formulas have the form

$$Q_1 y_1 Q_2 y_2 \cdots Q_n y_n \psi(x_1, \ldots, x_m, y_1, \ldots, y_n),$$

where $Q_1, \ldots, Q_n$ are quantifiers, $y_1, \ldots, y_n$ are the quantified variables, $x_1, \ldots, x_m$ are free variables, and ψ is a subformula that contains no quantifiers and no free variables other than $x_1, \ldots, x_m$. Because any formula can be transformed into a logically equivalent formula that is in prenex normal form, the results can be stated in terms of such formulas without loss of generality.

LEMMA A.6 (Ackermann)
Let $\varphi(x_1, \ldots, x_m)$ be an equality formula in prenex normal form, containing m free variables $x_1, \ldots, x_m$, n quantifiers, and l constant symbols $C_1, \ldots, C_l$. Let $k = l + m + n$. Let s_k be a state whose domain contains exactly k objects. Let s be a state whose domain contains more than k objects such that $s(C_i) = s_k(C_i)$ for all i, $1 \leq i \leq l$. Let $d_1, \ldots, d_m$ be objects in the domain of s_k and $e_1, \ldots, e_m$ be objects in the domain of s such that
1. $d_i = d_j$ if and only if $e_i = e_j$ for all i and j, $1 \leq i, j \leq m$.
2. $d_i = s_k(C_j)$ if and only if $e_i = s(C_j)$ for all i and j, $1 \leq i, j \leq l$.
Such objects can always be found, since $k \geq l + m$. It is then the case that $s_k[d_1/x_1, \ldots, d_m/x_m] \models \varphi(x_1, \ldots, x_m)$ if and only if $s[e_1/x_1, \ldots, e_m/x_m] \models \varphi(x_1, \ldots, x_m)$.

PROOF. The proof proceeds by induction on the number of quantifiers in $\varphi(x_1, \ldots, x_m)$. If $\varphi(x_1, \ldots, x_m)$ has no quantifiers, then the truth value of $\varphi(x_1, \ldots, x_m)$ in state $s_k[d_1/x_1, \ldots, d_m/x_m]$ depends entirely on the truth values of the atomic subformulas of $\varphi(x_1, \ldots, x_m)$. The same is true in state $s[e_1/x_1, \ldots, e_m/x_m]$. The atomic subformulas of $\varphi(x_1, \ldots, x_m)$ are of the forms $x_i = x_j$, $C_i = C_j$, $x_i = C_j$, and $C_j = x_i$. Because $d_i = d_j$ if and only if $e_i = e_j$, it follows that $s_k[d_1/x_1, \ldots, d_m/x_m] \models (x_i = x_j)$ if and only if $s[e_1/x_1, \ldots, e_m/x_m] \models (x_i = x_j)$. Likewise, since $s_k(C_i) = s_k(C_j)$ if and only if

$s(C_i) = s(C_j)$ and $d_i = s_k(C_j)$ if and only if $e_i = s(C_j)$, it follows that $s_k[d_1/x_1, \ldots, d_m/x_m] \models \psi$ if and only if $s[e_1/x_1, \ldots, e_m/x_m] \models \psi$, where ψ is any atomic subformula of the form $C_i = C_j$, $x_i = C_j$, or $C_j = x_i$. Because the truth values of the atomic formulas of $\varphi(x_1, \ldots, x_m)$ are the same in the two states, it follows that the lemma holds for any formula $\varphi(x_1, \ldots, x_m)$ containing no quantifiers.

For the induction step, suppose that the lemma holds for n quantifiers. Suppose further that $\varphi(x_1, \ldots, x_m)$ contains $n+1$ quantifiers. Then $\varphi(x_1, \ldots, x_m)$ is either of the form $\forall y\, \psi(x_1, \ldots, x_m, y)$ or $\exists y\, \psi(x_1, \ldots, x_m, y)$.

Suppose that $\varphi(x_1, \ldots, x_m)$ is of the form $\forall y\, \psi(x_1, \ldots, x_m, y)$ and that this formula is true in state $s_k[d_1/x_1, \ldots, d_m/x_m]$. Then, $s_k[d_1/x_1, \ldots, d_m/x_m, d/y] \models \psi(x_1, \ldots, x_m, y)$ for every object d in state s_k. Let e' be any object in state s. Let d' be an object in state s_k such that

1. $d' = d_i$ if and only if $e' = e_i$ for all i, $1 \leq i \leq m$.

2. $d' = s_k(C_i)$ if and only if $e' = s(C_i)$ for all i, $1 \leq i \leq l$.

Such an object can be found, since $k = l + m + (n+1) \geq m + 1$. Because $\psi(x_1, \ldots, x_m, y)$ contains only n quantifiers and $s_k[d_1/x_1, \ldots, d_m/x_m, d'/y] \models \psi(x_1, \ldots, x_m, y)$, it follows from the induction hypothesis, that $s[e_1/x_1, \ldots, e_m/x_m, e'/y] \models \psi(x_1, \ldots, x_m, y)$. Moreover, this holds for every object e' in state s, since the choice of e' was arbitrary. Hence, $\forall y\, \psi(x_1, \ldots, x_m, y)$ is true in state $s[e_1/x_1, \ldots, e_m/x_m]$.

Now suppose that $\forall y\, \psi(x_1, \ldots, x_m, y)$ is true in state $s[e_1/x_1, \ldots, e_m/x_m]$. Then $s[e_1/x_1, \ldots, e_m/x_m, e/y] \models \psi(x_1, \ldots, x_m, y)$ for every object e in state s. Let d' be any object in state s_k. Let e' be an object in state s such that

1. $d' = d_i$ if and only if $e' = e_i$ for all i, $1 \leq i \leq m$.

2. $d' = s_k(C_i)$ if and only if $e' = s(C_i)$ for all i, $1 \leq i \leq l$.

Such an object can be found, since s has more objects in its domain than s_k. By construction, $s[e_1/x_1, \ldots, e_m/x_m, e'/y] \models \psi(x_1, \ldots, x_m, y)$. Therefore, by the induction hypothesis, $s_k[d_1/x_1, \ldots, d_m/x_m, d'/y] \models \psi(x_1, \ldots, x_m, y)$. Since the choice of d' was arbitrary, it follows that $\forall y\, \psi(x_1, \ldots, x_m, y)$ is true in state $s_k[d_1/x_1, \ldots, d_m/x_m]$.

From this result and the previous result, we may conclude that $s_k[d_1/x_1, \ldots, d_m/x_m] \models \forall y\, \psi(x_1, \ldots, x_m, y)$ if and only if $s[e_1/x_1, \ldots, e_m/x_m] \models \forall y\, \psi(x_1, \ldots, x_m, y)$. By a similar argument it follows that $s_k[d_1/x_1, \ldots, d_m/x_m] \models \exists y\, \psi(x_1, \ldots, x_m, y)$ if and only if $s[e_1/x_1, \ldots, e_m/x_m] \models \exists y\, \psi(x_1, \ldots, x_m, y)$. Therefore, the lemma holds for any formula $\varphi(x_1, \ldots, x_m)$ containing $n+1$ quantifiers. Hence, by induction on n, the lemma holds for any number of quantifiers in φ. ∎

LEMMA A.7 (Ackermann's extension of Löwenheim's theorem)
Let $\varphi(x_1, \ldots, x_m)$ be an equality formula in prenex normal form containing m free variables $x_1, \ldots, x_m$, n quantifiers, and l constant symbols $C_1, \ldots, C_l$. Let $k = l + m + n$. Then, if $\varphi(x_1, \ldots, x_m)$ is true in a state with more than k objects, it is true in a state with exactly k objects.

PROOF. Suppose that $\varphi(x_1, \ldots, x_m)$ is true in a state s whose domain contains more than k objects. Let s_k be a state such that

1. The domain of s_k contains exactly k elements.

2. $s_k(C_i) = s(C_i)$ for all i, $1 \leq i \leq l$.

3. $s_k(x_i) = s(x_i)$ for all i, $1 \leq i \leq m$.

Such a state can be found, since $k \geq l + m$. Let $d_i = e_i = s_k(x_i) = s(x_i)$ for all i, $1 \leq i \leq m$. Then, by Lemma A.6, $s_k[s_k(x_1)/x_1, \ldots, s_k(x_m)/x_m] \models \varphi(x_1, \ldots, x_m)$ if and only if $s[s(x_1)/x_1, \ldots, s(x_m)/x_m] \models \varphi(x_1, \ldots, x_m)$. However, $s_k[s_k(x_1)/x_1, \ldots, s_k(x_m)/x_m] = s_k$ and $s[s(x_1)/x_1, \ldots, s(x_m)/x_m] = s$. Therefore, $s_k \models \varphi(x_1, \ldots, x_m)$ if and only if $s \models \varphi(x_1, \ldots, x_m)$. Hence, $\varphi(x_1, \ldots, x_m)$ is true in state s_k, since $\varphi(x_1, \ldots, x_m)$ is true in state s. Therefore, if $\varphi(x_1, \ldots, x_m)$ is true in a state whose domain contains more than k objects, it is also true in a state whose domain contains exactly k objects. ∎

THEOREM A.8 (Theorem 7.3)
There exist actions representable in ADL that do not have tidy progression operators.

PROOF. Consider the language that consists of the constant symbol 0 and the unary function symbol Add1. Let γ be the formula

$$\begin{aligned} &\forall x \big(\mathrm{Add1}(x) \neq 0\big) \\ &\wedge \forall x, y \big(\mathrm{Add1}(x) = \mathrm{Add1}(y) \rightarrow x = y\big) \\ &\wedge \forall x \big[x = 0 \vee \exists y \big(\mathrm{Add1}(y) = x\big)\big]. \end{aligned}$$

This formula defines Add1 to be the successor function of Peano Arithmetic [70, 8, 74]. Consequently, every state that satisfies γ has an infinite domain. Let Reset be the action defined in ADL as follows:

Reset ;;; *Set* $\mathrm{Add1}(x)$ *to be zero everywhere*

Update: $\mathrm{Add1}(x) \leftarrow 0$ for all x

This action has the effect of setting the interpretation of Add1 to be zero everywhere. From the semantics of ADL, Reset consists of the set of current-state/next-state pairs $\langle s, t \rangle$ such that

1. $\mathrm{Dom}(t) = \mathrm{Dom}(s)$.
2. For every variable symbol x, $t(x) = s(x)$.
3. $t(0) = s(0)$.
4. $t(\mathrm{Add1}) = \left\{ \langle d, e \rangle \in \mathrm{Dom}(s)^2 \mid s[e/y] \models (y = 0) \right\}$.

Assume that φ is a strongest postcondition of γ with respect to Reset. Because $\forall x \left(\mathrm{Add1}(x) = 0\right)$ is also a postcondition of γ with respect to Reset, φ must imply $\forall x \left(\mathrm{Add1}(x) = 0\right)$. Hence, φ is equivalent to $\varphi \wedge \forall x \left(\mathrm{Add1}(x) = 0\right)$. Let φ' be the formula obtained by transforming φ into prenex normal form and substituting 0 for every occurrence of a term of the form $\mathrm{Add1}(\tau)$. Then the formula $\varphi' \wedge \forall x \left(\mathrm{Add1}(x) = 0\right)$ is logically equivalent to φ, since $s(\mathrm{Add1}(\tau)) = s(0)$ for any term τ and any state s satisfying $\varphi \wedge \forall x \left(\mathrm{Add1}(x) = 0\right)$. Hence, φ' is also a postcondition of γ with respect to Reset.

Let $\langle s, t \rangle$ be an arbitrary pair of states in Reset such that $s \models \gamma$. The domain of s is therefore infinite. This implies that the domain of t is also infinite, since the domains of s and t are equal. In addition, t must satisfy φ'. By construction, φ' is an equality formula in prenex normal form. Therefore, by Lemma A.7, there exists a state t_k satisfying φ' whose domain contains exactly k objects, where k is the total number of constant symbols, free variables, and quantifiers in φ'. Let t_k be such a state, and let t'_k be the state obtained from t_k by setting the function denoted by Add1 to be equal to 0 everywhere. Then t'_k also satisfies φ' because Add1 does not appear in φ'. Furthermore, $t'_k \models \varphi$ because φ is logically equivalent to $\varphi' \wedge \forall x \left(\mathrm{Add1}(x) = 0\right)$ and t'_k satisfies both φ' and $\forall x \left(\mathrm{Add1}(x) = 0\right)$. Let ψ be the following formula:

$$\exists x_1 \cdots x_k \left[\forall y \left(y = x_1 \vee \cdots \vee y = x_k\right)\right].$$

This formula is true in those states whose domains contain k or fewer objects. In particular, $t'_k \models \psi$. Since t'_k satisfies both ψ and φ, it follows that $\varphi \not\models \neg\psi$. However, $\neg\psi$ is true in every state containing more than k objects. In particular, it is true in state t. Therefore, $\neg\psi$ is a postcondition of γ with respect to Reset. Since we assumed that φ is a strongest postcondition of γ with respect to Reset, it should follow that $\varphi \models \neg\psi$. However, $\varphi \not\models \neg\psi$. Hence, the assumption that φ is a strongest postcondition of γ with respect to Reset is incorrect. Since γ does not have a strongest postcondition with respect to Reset, we may conclude that there exist actions representable in ADL that do not have tidy progression operators, action Reset being one of them. ∎

Received 27 July 1993