Proving Goal Achievability

Fangzhen Lin
Department of Computer Science and Engineering
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong

Abstract

Given an action theory, a goal, and a set of initial states, we consider the problem of checking whether the goal is always achievable in every initial state of the set. To address this problem, we introduce a notion of reduction between sets of states, and show that if the set of the initial states can be reduced to one of its subsets, then the problem is equivalent to checking whether the goal is achievable in every initial state of the subset, provided that all the variables in the goal, if any, are existentially quantified, and that the preconditions and effects of the actions can be specified by quantifier-free formulas. We believe that this result provides an effective way of proving goal achievability, and illustrate it through some examples.

Introduction

A typical way of proving that a goal is achievable is to produce a plan to achieve it. Sometimes the very purpose of proving that a goal is achievable is to extract a plan from such a proof. However, when the achievability of a goal can be proved by some other, hopefully easier ways, we may want to verify that the given goal is indeed achievable before we actually attempt to find a plan to achieve it. The purpose of this paper is to provide such an alternative way, and show that sometimes, this alternative way is indeed easier.

To illustrate, consider the blocks world domain. Suppose that we want to know if we can always make block b on top of block a no matter what initial state we are given. One way to do this is to come up with a plan and prove that this plan will always achieve $on(a, b)$ no matter which initial state it is executed in.

Another way is to prove that this problem is equivalent to that of checking whether the goal is always achievable in every initial state *that has exactly two blocks in it, a and b*. The latter is almost a trivial problem as there are only five such states corresponding to the possible ways of arranging the two blocks, and whether the goal can be achieved in any of these states can be easily checked by either a depth-first or a breadth-first search as there are only five possible states to search for.

Of course, the question is how one can prove that the two problems are indeed equivalent. This is what this paper is about. We shall formulate certain conditions on the given action theory, the goal, and the class of initial states so that checking if the goal can be achieved in a set of initial state can be reduced to checking if the goal is achievable in a smaller subset of the initial states.

In addition to its possible application in planning, this result can be used to prove the correctness and termination of certain action programs with loops. For instance, consider the following Golog program (Levesque *et al.* 1997):

```
while $\neg clear(a)$ do
  $\pi(y)$ if $holding(y)$ then $putdown(y)$ endIf; (1)
  $\pi(y,z)$ if $clear(y) \wedge on(y,z) \wedge handempty$
      then $unstack(y,z)$ endIf
endWhile
```

Here π is the nondeterministic operator: $\pi(x)[\delta(x)]$ means nondeterministically pick an individual x, and for that x, perform $\delta(x)$. The correctness and termination of this program is equivalent to whether the goal $clear(a)$ can be achieved using actions $putdown(x)$ and $unstack(x, y)$ in every initial state of interest. Notice that the program (1) is what has been called a *universal plan* (Schoppers 1987) as it tells the agent how to achieve a given goal in every possible situation of concern.

This paper is organized as follows. We shall investigate the goal achievability problem in the situation calculus, so we first briefly review it in the next section. We then state and prove our main result, and illustrate its uses by some examples. We then discuss related work and conclude this paper.

Logical preliminaries

We assume a two-sorted first-order language L called *the domain language* below. The two sorts are $action$ for actions that can be performed by the agent, and $objects$ for things such as blocks, locations, trucks, etc. Fluents will be represented by predicates with $object$ arguments, and actions by functions from $object^n$ to $action$. Thus we consider only propositional fluents. Consequently, we assume that actions are the only proper functions in the language. In particular, there are no proper functions in the language whose ranges are of sort $object$. But there can be object constants. An example domain language for the blocks world is as follows:

- Two constants A and B of sort $object$ to denote two particular blocks.
- Fluents (predicates) $ontable(x, y)$, $clear(x)$, and $handempty$.
- Actions (functions) $stack(x, y)$ and $unstack(x, y)$.

Given such a domain language L, a corresponding situation calculus (McCarthy & Hayes 1969; Reiter 2001; Lin 2007b) is also a sorted first-order language. It has four sorts, and they are: $object$, $action$, $fluents$, and $situations$. The meaning of the sorts $object$ and $action$ are as before. The situation calculus inherits all the function symbols from the domain language L. Furthermore, for each predicate in L, there is a corresponding function in the situation calculus with the same name but with range $fluent$: if F is an n-ary predicate in L, then F is a n-ary function in the situation calculus with type $object^n \rightarrow fluent$. There is a constant S_0 of sort $situation$ denoting the initial situation, and inductively, a situation is either the initial situation or the resulting situation of applying an action in a situation. In addition to these functions, the situation calculus includes the following domain independent predicates and functions:

- The binary predicate H whose arguments are $fluent \times situation$. Informally, $H(f, s)$ means that the fluent f is true in the situation s. In the following, we shall sometimes write $H(p(\vec{t}), s)$ as $p(\vec{t}, s)$ for ease of reading.
- The binary predicate $Poss$ whose arguments are $action \times situation$. Informally $Poss(a, s)$ means that the action a is executable in s.
- The binary function do whose type is $action \times situation \rightarrow situation$. Informally, $do(a, s)$ is the situation resulted from performing the action a in the situation s. In the following, we extend this function to finite sequences of actions. If α is a finite sequence of actions, then we use $do(\alpha, s)$ to denote the situation resulted from performing α in s. Formally, $do(\alpha, s)$ is just a short hand standing for a situation term defined inductively as follows: $do([], s)$ is s, and if $do(\alpha, s)$ is the term S, then $do([\alpha; a], s)$ is the term $do(a, S)$, where $[\alpha; a]$ is the resulting sequence of appending a to the end of α.
- The binary predicate $\sqsubset$ in infix notation whose arguments are $situation \times situation$. Informally $s_1 \sqsubset s_2$ means that s_2 is the result of performing a finite non-empty sequence of actions in s_1.

If φ is a formula in the domain language L, and S a situation term in the situation calculus, then we denote by $\varphi[S]$ the situation calculus formula obtained from φ by replacing each (non-equality) atom A in it by $H(A, S)$. For instance,

$$(handempty \lor holding(A) \lor \exists x.on(x, A))[S_0]$$

is

$$H(handempty, S_0) \lor H(holding(A), S_0) \lor \exists x.H(on(x, A), S_0),$$

which can also be written as

$$handempty(S_0) \lor holding(A, S_0) \lor \exists x.on(x, A, S_0)$$

for ease of reading.

In the following, we call a formula in L a domain formula. If a domain formula does not mention any action terms, then we call it a *state formula*. A *state sentence* is a state formula that does not have any free variables.

A *basic action theory* $\mathcal{D}$ in the situation calculus is a set of axioms of the following form (Reiter 2001):

$$\Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0},$$

where

- Σ is the set of the following four foundational axioms:

  $$do(a, s) = do(a', s') \supset a = a' \land s = s',$$
  $$\forall P.[P(S_0) \land \forall a, s.P(s) \supset P(do(a, s))] \supset \forall s P(s),$$
  $$\neg s \sqsubset S_0,$$
  $$s \sqsubset do(a, s') \equiv s \sqsubseteq s',$$

  where $s \sqsubseteq s'$ is a shorthand for $s \sqsubset s' \lor s = s'$. The first axiom is the unique names assumption about the function do. The second one is an induction axiom in second-order logic specifying that the set of the situations is the smallest set that contains S_0 and closes under the do function. The last two axioms define the ordering $\sqsubset$ inductively.
- $\mathcal{D}_{ss}$ is a set of successor state axioms of the form:

  $$H(F(\vec{x}), do(a, s)) \equiv \gamma^+(a, \vec{x})[s] \lor H(F(\vec{x}), s) \land \neg\gamma^-(a, \vec{x})[s],$$

  where $\gamma^+(a, \vec{x})$ and $\gamma^-(a, \vec{x})$ are domain formulas. Informally, $\gamma^+(a, \vec{x})$ is the condition under which action a will make $F(\vec{x})$ true, and $\gamma^-(a, \vec{x})$ the condition under which it will make $F(\vec{x})$ false.
- $\mathcal{D}_{ap}$ is a set of action precondition axioms of the form:

  $$Poss(A(x_1, ..., x_n), s) \equiv \Pi(x_1, ..., x_n)[s],$$

  where $\Pi(x_1, ..., x_n)$ is a state formula whose free variables are among $x_1, ..., x_n, s$.
- $\mathcal{D}_{una}$ is the set of unique names axioms about actions.
- $\mathcal{D}_{S_0}$ is a set of sentences of the form $\varphi[S_0]$, where φ is a state sentence. This is the knowledge base for the initial situation S_0.

We say that an L-interpretation M is a *domain model* of $\mathcal{D}$ if it satisfies $\mathcal{D}_{una}$ and for each $\varphi[S_0] \in \mathcal{D}_{S_0}$, $M \models \varphi$. A domain model of $\mathcal{D}$ serves as an initial state for the action theory. Since a basic action theory completely describes the effects of each action in the domain and these actions are deterministic, once the initial state is fixed, all the successor states can be computed from the theory. Formally, given a domain model M and a basic action theory $\mathcal{D}$, we can extend M to a situation calculus model M_{sit} of $\mathcal{D}$ as follows:

- The domains Obj and Act for sorts $object$ and $action$ are the same as the domains for $object$ and $action$, respectively, in M.
- The domain for situations is defined inductively as: S_0 is a situation, and inductively, if S is a situation, then $do(A(\vec{u}), S)$ is also a situation, where A is an n-ary action and $\vec{u} \in Obj^n$. The function do is given Herbrand interpretation.

- $Poss$ is interpreted according to the action precondition axioms in $\mathcal{D}_{Poss}$.
- $\sqsubset$ is interpreted according to the foundational axioms.
- The truth value of H in S_0 is defined according to M: for each fluent f, $H(f, S_0)$ is true iff f is true in M. Inductively, for each n-ary fluent F, each $\vec{u} \in Obj^n$, the truth value of $H(F(\vec{u}), do(A(\vec{v}), S))$ is defined according to the successor state axioms in $\mathcal{D}_{ss}$.

Clearly, M_{sit} is a model of $\mathcal{D}$. In the following, we say that M_{sit} is the extension of M to the situation calculus under $\mathcal{D}$.

In the following, if M is a domain model of $\mathcal{D}$, and ξ is a sequence of ground actions, then we say that a model M' is the result of performing ξ in M if

- M' and M have the same domains, and interpret everything except fluents the same.
- for each fluent $F(\vec{x})$ and each variable assignment σ, $M', \sigma \models F(\vec{x})$ iff $M_{sit}, \sigma \models H(F(\vec{x}), do(\xi, S_0))$.

Notice that M' may or may not be a domain model as it may or may not satisfy $\mathcal{D}_{S_0}$.

The Main Theorem

We can now formally state the problem that we are interested in in this paper.

Given a basic action theory $\mathcal{D}$, a state sentence φ as the goal to achieve, and a domain model M of $\mathcal{D}$, we say that the goal φ is *achievable in* M (under $\mathcal{D}$) if $M_{sit} \models (\exists s) Exec(s) \wedge \varphi[s]$, where $Exec(s)$ is the formula:

$$\forall a, s'. do(a, s') \sqsubseteq s \supset Poss(a, s'). \qquad (2)$$

In the situation calculus, under the foundational axioms Σ, for each situation s, there is a sequence α of actions such that s is the result of performing α in the initial situation S_0. However, α may not be executable in S_0, and in this case s is like a "ghost situation" - it is there syntactically, but not meaningful. Our formula $Exec(s)$ defined above says that α, the sequence of actions that gets to s from S_0, is in fact a sequence of executable actions. Thus in words, the statement that φ is achievable in M means that there is a sequence α of actions such that α is executable in M and after it is executed, φ will be true in the end state.

If $\mathcal{M}$ is a class of domain models, then we say that the goal is *achievable in* $\mathcal{M}$ if it is achievable in every domain model in $\mathcal{M}$.

Our objective in this paper is to consider conditions on the given basic action theory $\mathcal{D}$, the goal φ, and the class $\mathcal{M}$ of domain models so that checking if φ is achievable in $\mathcal{M}$ can be done easily without having to produce an actual plan for doing this.

Our condition on the goal φ is simple to state: it can be any existentially quantified state sentence of the form $\exists\vec{x}\varphi$, where φ is quantifier-free. The conditions on $\mathcal{D}$ and $\mathcal{M}$ are more involved.

Ranking action theories

The complexity of an action theory depends on action precondition axioms $\mathcal{D}_{ap}$ and successor state axioms $\mathcal{D}_{ss}$. Arguably, the simplest action theories are those corresponding to STRIPS action theories where action preconditions are conjunctions of atoms, and the effects of actions are context independent. In the situation calculus, this means:

1. the action precondition axioms in $\mathcal{D}_{ap}$ have the form:

   $$Poss(A(\vec{x}), s) \equiv F_1(\vec{t_1}, s) \wedge \cdots \wedge F_k(\vec{t_k}, s),$$

   where each F_i is a fluent and $\vec{t_i}$ is a tuple of terms whose variables are among $\vec{x}$, and
2. for each action $A(\vec{x})$ and each fluent $F(\vec{y})$, the successor state axioms can entail a formula of the form

   $$F(\vec{x}, do(A(\vec{y}), s)) \equiv (F(\vec{x}, s) \wedge E_1) \vee E_2,$$

   where E_1 and E_2 are formulas that contains only equality atoms.

For instance, for the blocks world, we have

$$Poss(stack(x, y), s) \equiv holding(x, s) \wedge clear(y, s)$$

and

$$\begin{aligned} & on(x, y, do(stack(u, v), s)) \equiv \\ & \quad on(x, y, s) \vee (x = u \wedge y = v), \\ & clear(x, do(stack(u, v), s)) \equiv \\ & \quad (clear(x, s) \wedge x \neq v) \vee x = u. \end{aligned}$$

We can classify these simplest action theories as rank 0 action theories, and introduce a hierarchy of action theories of increasing complexity. For our purpose here, we introduce below a class of action theories that allow action preconditions to be defined by arbitrary quantifier-free formulas, and actions to have certain context dependency, and call it the class of rank 1 action theories.

Formally, a basic action theory $\mathcal{D}$ is said to be of rank 1 if it satisfies the following conditions:

1. For each action $A(\vec{x})$, its action precondition axiom is of the form

   $$Poss(A(\vec{x}), s) \equiv \Phi_A(\vec{x})[s], \qquad (3)$$

   where $\Phi_A(\vec{x})$ is a state formula that does not mention any quantifiers and whose free variables are among $\vec{x}$.
2. For each action $A(\vec{x})$, and each fluent $F(\vec{y})$,

   $$\mathcal{D}_{ss} \cup \mathcal{D}_{una} \models F(\vec{y})[do(A(\vec{x}), s)] \equiv \Phi_{F,A}(\vec{x}, \vec{y})[s] \quad (4)$$

   for a state formula $\Phi_{F,A}(\vec{x}, \vec{y})$ that does not mention any quantifiers and whose free variables are among $\vec{x}$ and $\vec{y}$.

Thus if $\mathcal{D}$ is of rank 1, then the truth value of a fluent $F(\vec{x})$ in a successor situation $do(A(\vec{y}), s)$ depends only on the truth values of the fluents on $\vec{x}$, $\vec{y}$, and constants. Clearly, all action theories of rank 0 are also of rank 1.

The main condition for an action theory to be of rank 1 is that the action preconditions and effects can be specified by quantifier-free formulas. So we can also call it a quantifier-free action theory. We could extend this "rank" hierarchy and introduce rank k action theories for any $k > 1$ by allowing some quantifications in the action precondition axioms and the action effect axioms. Since we are not going to deal with these action theories in this paper, we do not go into details about them here.

As an example, the following axiom about dropping an object that may or may not be fragile is allowed in a rank 1 action theory:

$$broken(x, do(drop(y, s))) \equiv \\ broken(x) \lor (y = x \land fragile(x, s)).$$

However, the following axiom about adding a value to a variable is not allowed as it has quantifiers in it:

$$value(x, u, do(add(y), s)) \equiv \\ \exists v_1, v_2.value(x, v_1, s) \land value(y, v_2, s) \land plus(v_1, v_2, u),$$

where $plus(v_1, v_2, u)$ means $u = v_1 + v_2$.

The following proposition captures a key property about rank 1 action theories.

Proposition 1 *Let $\mathcal{D}$ be a basic action theory of rank 1, and $\alpha_1, ..., \alpha_n$ a sequence of actions.*

1. *If φ is a state formula that does not have any quantifiers, then there is another state formula φ_0 that does not mention any quantifiers and does not have any variables other than those in φ or $\alpha_1, \cdots, \alpha_n$ such that*

$$\mathcal{D}_{una} \cup \mathcal{D}_{ss} \models \varphi[do([\alpha_1, \cdots, \alpha_n], S_0)] \equiv \varphi_0[S_0]. \quad (5)$$

2. *There is a state formula φ_1 that does not mention any quantifiers and does not have any variables other than those in φ or $\alpha_1, \cdots, \alpha_n$ such that*

$$\mathcal{D}_{una} \cup \mathcal{D}_{ss} \models Exec(do([\alpha_1, \cdots, \alpha_n], S_0)) \equiv \varphi_1[S_0]. \quad (6)$$

Proof:
1. This can be proved by induction on n, the number of actions in the sequence. $n = 0$ is trivial. The inductive case follows from (4). Effectively, we are using (4) to regress $F[do(\alpha, s)]$ to $\phi[s]$ for a state formula ϕ.

2. Again by induction on n. $n = 0$ is trivial. Inductively, $Exec(do(\alpha_n, do([\alpha_1, \cdots, \alpha_{n-1}], S_0))$ is equivalent to

$$Exec(do([\alpha_1, \cdots, \alpha_{n-1}], S_0)) \land \\ Poss(\alpha_n, do([\alpha_1, \cdots, \alpha_{n-1}], S_0)).$$

By (3), the second conjunct is equivalent to $\phi[do([\alpha_1, \cdots, \alpha_{n-1}], S_0)]$ for a state formula ϕ that do not have quantifiers and mentions only variables from $\alpha_1, \cdots, \alpha_n$. Thus by the first half of the proposition that we have already proved, and the inductive assumption, $Exec(do(\alpha_n, do([\alpha_1, \cdots, \alpha_{n-1}], S_0)))$ is equivalent to $\phi_1[S_0] \land \phi_2[S_0]$, for some state formulas ϕ_1 and ϕ_2 that do not mention any quantifiers and mention only variables from $\alpha_1, \cdots, \alpha_n$. ■

Model subsumption on universal sentences

We now consider the conditions that the class of initial states needs to satisfy. Recall that an initial state is formally a domain model in our formalism. So conditions on initial states are defined on domain models.

Definition 1 *We say that a model M_1* subsumes *another model M_2* on universal sentences of sort τ *if the following condition:*

$$\textit{if } M_2 \models \forall \vec{y}\varphi, \textit{ then } M_1 \models \forall \vec{y}\varphi$$

holds for every universal sentence of the form $\forall \vec{y}\varphi$, where $\vec{y}$ is a tuple of variables of sort τ and φ does not mention any quantifiers.

The following lemma is useful when proving this notion of subsumption between two models.

Lemma 1 *Let L be a first-order language, and M_1 and M_2 two structures of L with domains D_1 and D_2, respectively, for sort τ. If there is a mapping f from D_1 to D_2 such that for each predicate P (including equality), each tuple $\vec{t}$ of terms containing only variables of sort τ, and each variable assignment σ on D_1, $M_1, \sigma \models P(\vec{t})$ iff $M_2, f(\sigma) \models P(\vec{t})$, where $f(\sigma)$ is the variable assignment on D_2 such that $f(\sigma)(x) = f(\sigma(x))$ for each variable x of sort τ, then M_1 subsumes M_2 on universal sentences of sort τ.*

Proof: For any universal sentence $\forall \vec{y}\varphi$ of sort τ, if $M_1 \not\models \forall \vec{y}\varphi$, then there is a variable assignment σ such that $M_1, \sigma \models \neg\varphi$. Now let f be the function in the lemma, then $M_2, f(\sigma) \models \neg\varphi$. ■

Using this lemma, it is easy to see that if M_1 is a *submodel* of M_2 on sort τ, then M_1 subsumes M_2 on universal sentences of sort τ, where M_1 is a submodel of M_2 on sort τ if the following conditions hold:

- for sort τ, the domain of M_1 is a subset of the domain of M_2, and for other sorts, the domains of M_1 and M_2 are the same;
- the interpretation of each predicate and function in M_1 is the projection of its interpretation in M_2. In particular, if f is an n-ary function, then for each tuple $\vec{u}$ of elements in the domains of M_1, $f^{M_2}(\vec{u})$ must be in the domain of M_1.

The following proposition relates this notion of model subsumption with rank 1 action theories and goal achievability.

Proposition 2 *Let $\mathcal{D}$ be an action theory of rank 1, and M and M' two domain models of $\mathcal{D}$ such that M subsumes M' on universal sentences of sort $object$. For any state sentence $G = \exists \vec{x}\varphi$ such that φ does not mention any quantifiers, if G is not achievable in M', then G is not achievable in M either.*

Proof: Notice that the following sentence

$$\neg\exists s.Exec(s) \land \exists \vec{x}\varphi(\vec{x})[s] \quad (7)$$

is equivalent to the set of sentences of the following form

$$\forall \vec{y}.\neg(Exec(do(\xi, S_0)) \land \exists \vec{x}\varphi(\vec{x})[do(\xi, S_0)]),$$

where ξ is a finite sequence of actions, and $\vec{y}$ the tuple of variables in ξ. Without loss of generality, we assume that $\vec{x} \cap \vec{y} = \emptyset$. Thus the above sentence is equivalent to

$$\forall \vec{x}\vec{y}. \neg Exec(do(\xi, S_0)) \vee \neg\varphi(\vec{x})[do(\xi, S_0)].$$

By Proposition 1, under $\mathcal{D}$, this sentence is equivalent to

$$\forall \vec{x}\vec{y}. \neg\psi_1(\vec{y})[S_0] \vee \neg\psi_2(\vec{x}, \vec{y})[S_0]$$

for some state formulas ψ_1 and ψ_2 that do not have any quantifiers. This sentence is equivalent to

$$\forall \vec{x}\vec{y}. (\neg\psi_1 \vee \neg\psi_2)[S_0]. \qquad (8)$$

Now suppose G is not achievable in M'. This is equivalent to saying that the sentence (7) is true in M'_{sit}, and equivalently, (8) is true in M' for all ξ. Since M subsumes M', thus (8) is true in M for all ξ as well. So (7) is true in M_{sit}, therefore G is not achievable in M either. ■

The main result

We are now ready to put everything together. Given a basic action theory $\mathcal{D}$, we define a notion of *reduction* between two domain models inductively as follows:

- A model can be reduced to itself.
- Inductively, if M_1 can be reduced to M_2 and M_2 is subsumed by M_3 on universal sentences of sort $object$, then M_1 can be reduced to M_3.
- Inductively, if M_1 can be reduced to M_2 and M_3 is the result of performing a sequence of executable actions in M_2, then M_1 can be reduced to M_3.

Given two sets $\mathcal{M}$ and $\mathcal{M}'$ of domain models, we say that $\mathcal{M}$ can be *reduced* to $\mathcal{M}'$ if for every $M \in \mathcal{M}$ there is an $M' \in \mathcal{M}'$ such that M can be reduced to M'.

Theorem 1 *Let $\mathcal{D}$ be an action theory of rank 1, $\mathcal{M}$ a class of domain models of $\mathcal{D}$, and $\mathcal{M}'$ a subclass of $\mathcal{M}$. If $\mathcal{M}$ can be reduced to $\mathcal{M}'$, then for any goal $\exists\vec{x}\varphi$, where φ is a quantifier-free state formula, it is achievable in $\mathcal{M}$ iff it is achievable in $\mathcal{M}'$.*

Proof: Since $\mathcal{M}' \subseteq \mathcal{M}$, it is obvious that if a goal is achievable in $\mathcal{M}$ then it is achievable in $\mathcal{M}'$. Conversely, suppose a goal G of the form $\exists\vec{x}\varphi$ as in the theorem is not achievable in $\mathcal{M}$. Then there is an $M \in \mathcal{M}$ such that G is not achievable in M. By the assumption in the theorem, there is an $M' \in \mathcal{M}'$ such that M can be reduced to M'. We show by induction that G is not achievable in M' either. The base case is trivial. For the inductive case, we need to show that

1. if M_1 is subsumed by M_2 on universal sentences of sort $object$ and G is not achievable in M_1, then it is not achievable in M_2 either: this follows from Proposition 2.
2. if M_1 is the result of performing a sequence ξ of executable actions in M_2, and G is not achievable in M_2, then it is not achievable in M_1 either: if G is achievable in M_1, say by the sequence θ of executable actions, then doing θ after ξ would achieve G in M_2.

Examples

Our first example is again about the blocks world. We are interested in finding out which goals can be achieved if the agent is only allowed to perform two kinds of actions: $putdown(x)$ (put down block x on the table, provided the robot is holding it) and $unstack(x, y)$ (pick up block x from block y, provided x is on y and clear, and that the robot's hand is empty).

Since a goal needs to be a state sentence, we assume that there is a constant a in our language so that we can consider goals such as $clear(a)$ and $ontable(a)$. To consider goals such as $on(a, b)$, we need to have two constants in the language. We shall come back to this later.

The action precondition axioms and successor state axioms for the blocks world are well-known and can be found in many places (e.g. (Reiter 2001; Lin 2007b)). We assume that $\mathcal{D}_{S_0} = \emptyset$, and define the set of legal initial states model-theoretically as below.

The class of legal initial states is captured by the class $\mathcal{M}$ of the following domain models:

- The object domain is a finite set of objects called blocks that contains at least one block "B_0" which is the denotation of the constant a.
- Finite number of towers with the top one clear, and the bottom one on the table. The robot's hand may or may not be empty, and if it is not empty it is holding exactly one block. A block must be at exactly one location: either in the robot's hand, on the table, or on another block.
- The action domain is given Herbrand interpretation, thus will satisfy the unique names axioms $\mathcal{D}_{una}$.

Now let $\mathcal{M}_0$ be the domain models in $\mathcal{M}$ that have only B_0 in their object domain. We claim that $\mathcal{M}$ can be reduced to $\mathcal{M}_0$, thus by Theorem 1, whether a goal of the form $\exists\vec{x}\varphi$ can be achieved in $\mathcal{M}$ can be checked using $\mathcal{M}_0$ which contains only two models.

Let $M \in \mathcal{M}$. Suppose that in addition to B_0, M has another block in its domain. If the robot is holding a block, say x in M, perform $putdown(x)$. If there is a tower of at least 2 blocks, say y and z with z on the top and on y, then do $[unstack(y, z), putdown(y)]$. Let's call the resulting domain model M'. Clearly, $M' \in \mathcal{M}$ and M can be reduced to M' (M' is M when all the blocks are on the table in M). Now let M'' be the submodel of M' on the set of blocks that are in the same tower as B_0. It can be seen that $M'' \in \mathcal{M}$. i.e. a legal state. Since M'' is a submodel of M' on sort $object$, thus M'' subsumes M' on universal sentences of sort $object$. So M' can be reduced to M''. Thus M can be reduced to M''. Furthermore, the number of blocks in M'' is smaller than that in M. If M'' has more than one block, then we can apply the same procedure to M''. Since our notion of "reduction" is transitive, by induction, M can be reduced to a domain model that has B_0 the only block in its domain, i.e. in $\mathcal{M}_0$.

There are only two elements in $\mathcal{M}_0$, one in which $clear(a)$ holds and the other where $holding(a)$ holds. We can then easily check that $clear(a)$, $ontable(a)$, $\neg holding(a)$, and $handempty$ are achievable in $\mathcal{M}_0$ thus also in $\mathcal{M}$, but $\neg clear(a)$, $\neg ontable(a)$, $holding(a)$, and $\neg handempty$ are not.

Given this, it follows that the following Golog program that we mentioned earlier:

```
while $\neg clear(a)$ do
    $(\pi y)$ if $holding(y)$ then $putdown(y)$ endIf;
    $\pi(y,z)$ if $clear(y) \wedge on(y,z) \wedge handempty$ then
            $unstack(y,z)$ endIf
endWhile
```

can be executed to a terminating state that satisfies $clear(a)$.

We mentioned earlier that if we want to consider goals like $on(a,b)$, we have to consider a language with two constants, a and b. What we have done above can be carried over to this case straightforwardly: the subclass $\mathcal{M}_0$ will now be all those domain models with two blocks, say B_0 and B_1, and when we consider submodels, the object domains would need to include all the blocks that are either in the same tower as B_0 or in the same tower as B_1. Again the achievability of a goal of the form $\exists\vec{x}\varphi$ in $\mathcal{M}$ is equivalent to the achievability of the goal in $\mathcal{M}_0$, which now has five models: one in which both blocks are on the table, one $on(a,b)$ is true, one $on(b,a)$ is true, one $holding(a)$ is true, and one $holding(b)$ is true. Checking these models one by one, it can be seen that $on(a,b)$ is not achievable in all states, but $clear(a) \wedge clear(b)$ is achievable, so is $clear(a) \wedge ontable(b)$, and so on.

As another example, consider the chopping tree example (Sardina *et al.* 2004; Levesque 2005). In this domain, there is a tree that can be chopped down, but the agent does not know how many times that she needs to chop at it in order to bring it down. Levesque (2005) axiomatized this problem using a sensing action that can inform the agent if the tree is down, and his planner can generate a program like the following one for achieving the goal:

```
sense;
while tree_not_down
 chop;
 sense;
endwhile
```

In general, programs generated by Levesque's planner are not guaranteed to be correct. One possible application of our work is for proving the correctness of these programs. The problem, however, is that we do not have sensing actions here. One possible solution is to find a systematic way of translating programs with sensing actions to ones without, perhaps by turning sensing fluents into ordinary ones together with new actions that change these ordinary fluents accordingly. We don't know yet how to do this generally, but we can illustrate the idea using the chopping tree example.

Instead of a sensing action that can tell the agent if the tree is down, we introduce a $size$ fluent that denotes the number of chops that is needed to bring down the tree, and assume that the agent does not know the value of this fluent in the initial situation. Thus the problem now is to come up with a program that will bring down the tree no matter what the initial value of $size$ is. The following formulation is one way to make this precise.

We use three fluents: $down$ (the tree is down), $size(n)$ (the size of the current tree is n), and a relation $succ(m,n)$ ($m = n+1$). There is one action $chop(m,n)$ which decreases the size of the tree from m to n, provided $n = m-1$ (thus it just decreases the size by one). The following are the action precondition axioms and successor state axioms:

$$\begin{aligned}
&Poss(chop(m,n),s) \equiv \\
&\qquad succ(m,n) \wedge \neg down(s) \wedge size(m,s), \\
&size(x, do(chop(m,n),s)) \equiv x = n, \\
&down(do(chop(m,n),s)) \equiv m = 1, \\
&succ(m',n',do(chop(m,n),s)) \equiv succ(m',n',s).
\end{aligned}$$

Now consider the following program:

```
while $\neg down$ do
    $\pi(m,n)$ if $succ(m,n) \wedge size(m)$ then $chop(m,n)$
endWhile
```

Again, we can reduce the problem of termination and correctness of this program to goal achievability.

Let $\mathcal{D}_{S_0} = \emptyset$, and the set $\mathcal{M}$ of legal initial states be the domain models that satisfy the following properties:

- The domain D is $\{1, 2, \cdots, n\}$ for some $n \geq 1$.
- The constant 1 is interpreted as the number 1 in D.
- $succ(m,n)$ iff $m = n+1$.
- There is exactly one $m \in D$ such that $size(m)$ holds.
- $down$ does not hold.

Let M_0 be the model in $\mathcal{M}$ whose domain is $\{1\}$ (there can be only one such model), and $\mathcal{M}_0 = \{M_0\}$. We show that $\mathcal{M}$ can be reduced to $\mathcal{M}_0$.

Let M be a domain model in $\mathcal{M}$. Let M' be the submodel of M on $\{1, ..., m\}$, where m is such that $size(m)$ holds in M. If $m = 1$, then $M' \in \mathcal{M}_0$. If $m > 1$, then perform $chop(m, m-1)$ in M', and call the resulting model M''. Clearly, $M'' \in \mathcal{M}$. Now let M''' be the submodel of M'' on $\{1, ..., m-1\}$. If $m - 1 = 1$, then $M'' \in \mathcal{M}_0$, otherwise continue this process, and eventually M will be reduced to the model in $\mathcal{M}_0$.

Thus for any goal of the form $\exists\vec{x}\varphi$, it is achievable in $\mathcal{M}$ iff it is achievable in $\mathcal{M}_0$. In particular, $down$ is achievable, thus the above program can be executed to a terminating state. However, the goal $\exists x.size(x) \wedge succ(x,1)$ is not achievable.

For these two problems, we have been able to reduce the set of initial states to such a subset that both the size of the subset and the size of the states in the subset are very small. We now look at an example for which while the reduced subset is still large, but whether the goal is achievable in any of the states in the subset is trivial to decide.

Consider the omelette's problem (Bonet & Geffner 2001; Levesque 2005) where the agent needs to get a certain number of good eggs in a bowl to make an omelette. The agent is given a basket of eggs, some of them good and others bad. The agent can only know if any of the eggs is good or not by breaking it into a container, say a cup. If the egg is good, then the agent transfers the egg from the cup to the bowl. If the egg is not good, then the agent just throws away the egg in the cup.

Here we assume that the eggs are lined up to be used according to a linear order. A Golog program for solving this problem is then as follows:

```
while $\neg om(n)$ do
  $\pi(x,y)$ if $currentEgg(x) \wedge next(x,y) \wedge cupEmpty$
      then $break(y)$ endIf
  $\pi(x,m,k)$ if $cup(x) \wedge good(x) \wedge om(m) \wedge succ(m,k)$
      then $transferToBowl(x,m,k)$ endIf
  $(\pi x)$ if $cup(x) \wedge \neg good(x)$ then $emptyCup(x)$ endIf
endWhile
```

where $om(n)$ means that there are currently n good eggs in the bowl, $next(x,y)$ that the egg y is the next one after x to be used, and $cup(x)$ that the egg x is broken and in the cup. Action $break(y)$ breaks egg y into the cup provided y is the next egg to be used and the cup is empty, $transferToBowl(x,m,k)$ transfers the egg x from the cup to the bowl and add by 1 the number of eggs in the bowl, provided the egg x is good and in the cup, and $emptyCup(x)$ simply dumps the egg. The effects of these actions can be specified similarly as in the tree chopping example above.

The crucial part is the specification of the possible initial states. Recall that our goal is to reduce the given class of initial states to a smaller subclass. If we always start with the empty bowl, $om(0)$, we can only remove bad eggs from the domain, but never any good eggs as adding any good egg into the bowl will make $om(0)$ false, thus resulting in a state that is not in the class of initial states. Now assume that the class of initial states is the set of domain models satisfying the following conditions:

- There is a unique i such that $om(i)$ holds, and for this i, $i \leq n$ (n is a constant in the language denoting the number of good eggs needed to make the omelette).
- $succ(x,y)$ if $y = x + 1$.
- There is a finite number of eggs and $next(x,y)$ is a linear ordering on them, i.e. $next(x,y)$ is functional on both arguments, and non-circular.
- $currentEgg(x)$ holds for exactly one x.
- $cupEmpty$ holds iff $\neg\exists x.cup(x)$.

This class can be reduced to the subclass of states that satisfy the following condition:

- either $om(n)$ holds or
- $om(i)$ holds for some $i < n$ and there are no eggs in the domain.

Clearly, the goal is not achievable in any state where there is no more eggs available but $om(n)$ is not true. Ruling out these states corresponds to requiring that in the original class of initial states, if $om(i)$ holds then there are at least $n - i$ good eggs in the domain.

Related work

Broadly speaking, this work is related to planning and reasoning about action. The closely related work includes Reiter's pioneering work on proving state properties in the situation calculus using induction (Reiter 1993). Reiter showed how certain state properties such as a universally quantified state constraint can be formally proved in the situation calculus using induction, and gave some examples on how this can be used to formally show that certain goal is not achievable. While in general proving whether a goal can be achieved needs induction, what we have done here is to show that for some special cases, the problem can be reduced to checking whether a goal can be achieved in some small domains which is then done by model checking.

This work is also closely related to, in fact, motivated by the notion of finitely verifiable classes of sentences that we proposed earlier (Lin 2007a). A class of sentences is finitely verifiable if there is a finite set of models such that a sentence in the class is a theorem iff it is true in all models of the set. Given a basic action theory $\mathcal{D}$ and a class $\mathcal{M}$ of models, we can say that a goal G is a "theorem" if it is achievable in $\mathcal{M}$ under $\mathcal{D}$. Thus our main result says that if $\mathcal{M}$ can be reduced to a finite subset $\mathcal{M}_0$, then checking whether G is a "theorem" is equivalent to checking whether G is "true" (achievable) in every model in $\mathcal{M}_0$. This is a very useful analogy as $\mathcal{M}$ is often large, even infinite, and the objective is to reduce it to a small subset $\mathcal{M}_0$.

Concluding remarks

We have proved a technical result that can be used to check the achievability of a goal in a class of initial states under an action theory, and showed its usefulness through some examples. While we have formulated the problem and our results in the situation calculus, the same can be done using other action formalisms. For instance, in STRIPS and ADL formalisms, it is easy to define similar notions like states, executable sequences of actions in a state, goals, and goal achievability in a state and in a class of states. There has also been much work on relating the situation calculus to STRIPS and ADL, e.g. (Lifschitz 1986; Pednault 1989; Lin & Reiter 1997)), and it is clear that our rank 1 action theories correspond to ADL action domains where the precondition, add, and delete lists do not have quantifiers.

Acknowledgments

I want to thank the anonymous reviewers for their helpful comments on an earlier version of this paper. This work has been supported in part by HK RGC CERGs 616707 and 616208.

References

Bonet, B., and Geffner, H. 2001. Gpt: a tool for planning with uncertainty and partial information. In *Proceedings of IJCAI-01 Workshop on Planning with Uncertainty and Partial Information*, 82–87.

Levesque, H.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. 1997. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming, Special issue on Reasoning about Action and Change* 31:59–84.

Levesque, H. J. 2005. Planning with loops. In Kaelbling, L. P., and Saffiotti, A., eds., *IJCAI*, 509–515. Professional Book Center.

Lifschitz, V. 1986. On the semantics of STRIPS. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, 1–9. Morgan Kauffmann Publishers, Inc. June 30–July 2, Timberline, Oregon.

Lin, F., and Reiter, R. 1997. How to progress a database. *Artificial Intelligence* (92)1-2:131–167.

Lin, F. 2007a. Finitely-verifiable classese of sentences. In *Proceedings of the 2007 AAAI Spring Symposium on Logical Formalization of Commonsense Reasoning*. http://www.ucl.ac.uk/commonsense07/.

Lin, F. 2007b. Situation calculus. In van Harmelen, F.; Lifschitz, V.; and Porter, B., eds., *Handbook of Knowledge Representation*. Elsevier.

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., and Michie, D., eds., *Machine Intelligence 4*. Edinburgh: Edinburgh University Press. 463–502.

Pednault, E. P. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, 324–332. Morgan Kaufmann Publishers, Inc.

Reiter, R. 1993. Proving properties of states in the situation calculus. *Artificial Intelligence* 64:337–351.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press.

Sardina, S.; Giacomo, G. D.; Lespérance, Y.; and Levesque, H. J. 2004. On the semantics of deliberation in indigolog - from theory to implementation. *Annals of Mathematics and Artificial Intelligence* 41(2-4):259–299.

Schoppers, M. 1987. Universal plans for reactive robots in unpredictable environments. In *Proceedings of IJCAI 1987*, 1039–1046.