Precondition Control and the Progression Algorithm

Alfredo Gabaldon
Department of Computer Science
University of Toronto
alfredo@cs.toronto.edu

Abstract

We consider the problem of planning with declarative search control in the framework of the situation calculus. In particular, we are concerned with forward-chaining planning with search control expressed by linear temporal logic formulas in the style of Bacchus & Kabanza's TLPlan system. We introduce a procedure for extracting conditions which can then be evaluated on plan prefixes for enforcing search control. These conditions are situation calculus formulas whose entailment can be checked by regression and can be used as further preconditions of actions in nonMarkovian action theories and in some cases in classical Markovian theories through a transformation procedure. We show that these conditions eliminate exactly the same plan prefixes as the Progression algorithm of the TLPlan system.

Introduction

Due to the high computational complexity of the classical planning problem, many strategies have been proposed to make the problem more tractable. Some of these strategies consist in taking advantage of the availability of knowledge about the structure of a particular domain. Prominent work in this direction includes: Hierarchical Task Network (HTN) planning systems (Sacerdoti 1974; Wilkins 1988; Erol, Hendler, & Nau 1996), which use procedural knowledge; SAT-solver based planning systems such as SATPlan (Kautz & Selman 1998), which use declarative constraints; and the forward-chaining planning systems TLPlan (Bacchus & Kabanza 2000) and TALPlanner (Kvarnström & Doherty 2000) which are based on Bacchus & Kabanza's proposal to provide the planning system with search control in the form of linear temporal logic formulas. These formulas can be seen as additional properties that a plan must have and give the system some degree of direction in its search for a plan. These properties may be in terms of any state the domain would go through if the plan were executed, and so linear temporal logic is a natural choice for expressing them.

In the tire world, for example, in searching for a plan to change a flat tire, one could use a piece of search control saying: "tools are not put away until they are no longer needed." During planning, such advice can be used to eliminate sequences of actions that will lead to dead-ends or to suboptimal plans. This effectively reduces the search space for the planning system and hence there is a computational advantage which has indeed been verified empirically in experiments and in the AIPS planning competitions. Planners using search control have shown several orders of magnitude superior performance in multiple planning domains. In many of these domains, problems of size formerly beyond the capabilities of any planning system have been solved using control.

The TLPlan and TALPlanner systems use search control through a temporal formula *progression algorithm*. This algorithm takes a state and a temporal logic formula, and computes a new formula. When the planner picks a new candidate action to append it to the current sequence, it computes the state that results from executing this action and then calls the progression procedure. This then checks that the new state satisfies the current formula, and returns a new formula that is to be checked next.

With the purpose of further computational improvement, it has been suggested that some forms of control knowledge are better incorporated as further preconditions of actions (Bacchus & Ady 1999; Kvarnström & Doherty 2000; Rintanen 2000; Kvarnström 2002; Gabaldon 2003). Preliminary experiments (Bacchus & Ady 1999; Kvarnström 2002) indicate that control as preconditions does indeed lead to a speed up. However, a systematic procedure for deriving precondition control from general temporal logic formulas has not been developed so far. One of our goals in this paper is to move toward filling this gap.

Using the situation calculus (McCarthy 1963) as our formal framework, specifically Reiter's situation calculus action theories (Reiter 1991; 2001), we look at planning with search control of the form used by the TLPlan and TALPlanner systems. First, we define a set of constraints that are obtained from a search control formula and which can be used as precondition control. These constraints may refer to any past situation in addition to the current one, so we use a recent extension of Reiter's action theories in which the *Markov property* is not assumed (Gabaldon 2002). We then adapt the progression algorithm to our logical framework and show that the aforementioned constraints and the progression algorithm eliminate exactly the same plan prefixes. Finally, we discuss how by applying the compilation technique presented in (Gabaldon 2003), we can take a non-Markovian action theory with precondition control and obtain a Markovian one that has the search control "compiled" into it. Put together, given a linear temporal logic formula, this is a procedure for obtaining precondition control from it that is equivalent to applying the control through the progression algorithm.

Basic Action Theories

We axiomatize planning domains in the situation calculus. This sorted second-order language has three basic components: actions, situations, and fluents. A distinguished set of function symbols is used to form actions: terms such as $open(x)$. Situations are terms that denote sequences of actions and represent possible "histories" of the domain. The initial situation is denoted by the constant S_0 and other histories are constructed by means of the do function which maps an action and a situation into a situation. The term $do(open(box), do(unlock(box), S_0))$ is an example. Fluents denote properties of the domain that change as a result of executing actions. They can be functional or relational and they all have a situation as an argument in the last position. A real valued function $velocity(x, s)$ and a relation $locked(x, s)$ are examples of fluents. We only use relational fluents in this paper.

A *basic action theory* (Pirri & Reiter 1999; Reiter 2001) in this language includes four domain independent axioms which axiomatize situations and the situation predecessor relation $\sqsubset$. Since in our framework situations are sequences of actions, that a situation s' precedes a situation s, i.e., $s' \sqsubset s$, means that the sequence s' is a prefix of sequence s.

The domain dependent axioms in a basic action theory are the following:

1. For each action function $A(\vec{x})$, an *action precondition axiom* of the form:
$$Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s).$$
2. For each fluent $F(\vec{x}, s)$, a *successor state axiom* of the form:
$$F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s).$$
3. Unique names axioms for actions. For instance: $open(x) \neq pickup(x)$.
4. Axioms describing the initial situation of the world: a finite set of sentences whose only situation term may be the constant S_0.

Action theories as defined in (Pirri & Reiter 1999; Reiter 2001) are Markovian: the preconditions of an action and its effects depend only on the situation where the action is to be performed. This property is realized in an action theory through the requirement that formulas $\Pi_A(\vec{x}, s)$ and $\Phi_F(\vec{x}, a, s)$ not mention relation $\sqsubset$ nor any situation term other than variable s.

A nonMarkovian action theory (Gabaldon 2002) allows, under certain restrictions, that these formulas refer to past situations, and to situations in a bounded future, with respect to the current situation s.

For example, a nonMarkovian precondition axiom for a computer system log-on action that allows only two attempts could be:

$$\begin{array}{l} \quad Poss(log\_on(usr, pwd), s) \equiv \\ verified(usr, pwd, s) \land \neg(\exists s_1, s_2)\{s_2 \sqsubset s_1 \sqsubseteq s \land \\ denied(usr, s_1) \land denied(usr, s_2) \land \\ (\forall s_3)(s_2 \sqsubset s_3 \sqsubset s) \supset \neg loggedOn(usr, s_3)\}. \end{array}$$

Before we proceed, a word on notation: while lower case letter arguments in predicate and function symbols denote variables, for example a for actions, s for situations, x for objects; we will use α to denote arbitrary but explicit action terms such as $move(x, y)$, and σ for situation terms such as $do(move(x, y), s)$. All symbols may have sub/superscripts. Variables that appear free are universally quantified unless stated otherwise.

Logical specification of the planning task

Perhaps the earliest formal specification of the planning problem is due to Green (1969) and was postulated in the situation calculus. Given a background axiomatization of the planning domain, which in our case would be an action theory $\mathcal{D}$, and a formula $G(s)$ representing the goal as a property that must hold in the final situation s, the planning problem is specified as the problem of proving the following entailment:

$$\mathcal{D} \models (\exists s).executable(s) \land G(s)$$

where

$$executable(s) \stackrel{\text{def}}{=} (\forall a, s').do(a, s') \sqsubseteq s \supset Poss(a, s').$$

Put in words, planning is defined as the problem of proving the existence of a plan s that is executable, i.e., the preconditions of each action are satisfied when it is executed, and such that the goal holds at the end.

We are concerned with planning using search control, so let us start with a specification of this problem based on Green's. As we mention in the introduction, the type of search control we are interested in using consists in properties of sequences. Of course the goal can be considered a special kind of sequence property that happens to refer only to the final situation s. In general, the search control is a property that refers to any situation lying between an initial situation s and a final s'. Denote this by $SC(s, s')$.

Now, even if s is a plan in the sense that $G(s)$ holds, it is not necessarily the case that $SC(S_0, s)$ hold. If a plan is found following the search control, what we have instead of $SC(S_0, s)$ is that for a situation s' equal to or succeeding s, $SC(S_0, s')$ holds.

Formally, we will use the following specification of planning with a classical goal[1] $G(s)$ and a search control formula $SC(s', s)$:

[1]Temporally extended goals (Bacchus & Kabanza 1998) are similar to the control formulas $SC(s', s)$ and can be incorporated in a straight forward manner.

$$\mathcal{D} \models (\exists s).executable(s) \wedge G(s) \wedge \\ (\exists s')\{s \sqsubseteq s' \wedge SC(S_0, s')\}. \quad (1)$$

Control Expressions

Let us introduce the syntax of the expressions we will use for search control. These expressions will basically be obtained from situation calculus formulas by removing the situation arguments from atoms and by adding linear temporal logic operator notation.

Given a situation calculus formula that does not mention relation $\sqsubset$ nor equality between situations, a *situation-suppressed expression* is obtained by removing the situation argument from all fluent and $Poss$ atoms in the situation calculus formula.

For a situation suppressed expression ϕ, we will use $\phi[\sigma]$ to denote the situation calculus formula obtained by placing σ as the situation argument in all the fluent and $Poss$ atoms in ϕ. For instance, if ϕ is $(\exists x).on(x, A) \wedge clear(x)$, where on and $clear$ are fluents, then $\phi[do(a, s)]$ stands for $(\exists x).on(x, A, do(a, s)) \wedge clear(x, do(a, s))$. Furthermore, for a situation-suppressed expression ϕ, $\phi[s', s]$ denotes $\phi[s']$.

For temporal expressions, we will borrow the temporal logic symbols $\circ$ (next), $\mathbf{U}$ (until), $\Box$ (always), and $\Diamond$ (sometime). We will use them with situation suppressed expressions to form control expressions.

Definition 1 *The control expressions are the smallest set of expressions such that*

1. *a situation suppressed expression is a control expression;*
2. *if ϕ and ψ are control expressions, then so are $\circ\phi$, $\phi \mathbf{U} \psi$, $\Box\phi$, $\Diamond\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, $\neg\phi$, $(\exists v)\phi$, $(\forall v)\phi$.*

As in the case of situation suppressed expressions, for a control expression ϕ, $\phi[s', s]$ is an abbreviation for a situation calculus formula. Intuitively, by $\phi[s', s]$ we mean that the temporal expression ϕ holds over the sequence s which starts at s'. For instance $\Box p[s', s]$ intuitively means that p is true in every situation between s' and s, inclusive.

Obviously, these control expressions are not meant to capture the semantics of temporal modalities, since the semantics of the temporal logic operators is defined in terms of sequences (of worlds) that are infinite. We introduce these expressions for two reasons: 1) they are a convenient notation for writing this type of search control and 2) it facilitates an analysis of the relationship between our description of planning with search control in the situation calculus and planning with search control through the progression algorithm.

The precise definition of the abbreviations for the temporal expressions is the following:[2]

[2]We use the following abbreviations throughout:
$(\exists s_1 : s' \sqsubseteq s_1 \sqsubseteq s)W \overset{\text{def}}{=} (\exists s_1)\{s' \sqsubseteq s_1 \wedge s_1 \sqsubseteq s \wedge W\}$
and
$(\forall s_1 : s' \sqsubseteq s_1 \sqsubseteq s)W \overset{\text{def}}{=} (\forall s_1)\{[s' \sqsubseteq s_1 \wedge s_1 \sqsubseteq s] \supset W\}$.

$$\circ\phi[s', s] \overset{\text{def}}{=} (\exists a).do(a, s') \sqsubseteq s \wedge \phi[do(a, s'), s]$$

$$\phi \mathbf{U} \psi[s', s] \overset{\text{def}}{=} (\exists s_1 : s' \sqsubseteq s_1 \sqsubseteq s)\{\psi[s_1, s] \wedge \\ (\forall s_2 : s' \sqsubseteq s_2 \sqsubset s_1)\phi[s_2, s]\}$$

$$\Diamond\phi[s', s] \overset{\text{def}}{=} (\exists s_1 : s' \sqsubseteq s_1 \sqsubseteq s)\phi[s_1, s]$$

$$\Box\phi[s', s] \overset{\text{def}}{=} (\forall s_1 : s' \sqsubseteq s_1 \sqsubseteq s)\phi[s_1, s]$$

Just as the temporal logic connectives $\Box$ and $\Diamond$ can be defined in terms of $\mathbf{U}$, a similar relationship holds in the case of the above expressions:

$$\Diamond\phi[s', s] \equiv True \mathbf{U} \phi[s', s]$$

and

$$\Box\phi[s', s] \equiv \neg\Diamond\neg\phi[s', s].$$

Forward-chaining planning with search control

Forward-chaining planning systems build plans by incrementally adding actions to an initially empty sequence. These systems use the search control knowledge to eliminate some of the partial sequences that, according to the control knowledge, cannot be extended into a full plan or will result in a plan which is of lower quality in some sense.

In terms of the situation calculus, suppose that a new action α is added to a sequence σ'. If the resulting sequence $\sigma = do(\alpha, \sigma')$ satisfies the goal, i.e., $\mathcal{D} \models G(\sigma)$, then we are done. Otherwise, σ can be extended into a full plan iff

$$\mathcal{D} \models (\exists s).\sigma \sqsubset s \wedge G(s) \wedge \\ (\exists s').s \sqsubseteq s' \wedge SC(S_0, s').$$

Thus the new sequence σ must satisfy the following:

$$\mathcal{D} \models (\exists s').\sigma \sqsubset s' \wedge SC(S_0, s'). \quad (2)$$

The problem of verifying this condition on σ is a problem no different to the general planning problem itself. This rules out checking the existence of a situation s' as specified in (2).

So we compromise and weaken the requirement. All the control checking will be done on the partial plan prefix σ instead of on a future situation s' as in (2). That is, we will only check whether σ satisfies some properties that are necessarily true when condition (2) holds. We define such properties for the temporal modalities introduced in the previous section.

Plan Prefix Control Conditions

Let us consider the formula for the temporal expression with $\mathbf{U}$. Suppose that the search control is dictated by the expression $\phi \,\mathbf{U}\, \psi$.

The instance of condition (2) for this control expression, on a prefix σ is the following:

$$\begin{aligned}&(\exists s').\sigma \sqsubset s' \wedge \\ &\quad (\exists s_1 : S_0 \sqsubseteq s_1 \sqsubseteq s')\{\psi[s_1, s'] \wedge \\ &\qquad (\forall s_2 : S_0 \sqsubseteq s_2 \sqsubset s_1)\phi[s_2, s']\}\end{aligned}$$

From this, we can derive two cases based on whether s_1 is a situation succeeding or preceding σ:

$$\begin{aligned}&(\exists s')\sigma \sqsubset s' \wedge (\exists s_1 : \sigma \sqsubset s_1 \sqsubseteq s')\{\psi[s_1, s'] \wedge \\ &\qquad (\forall s_2 : S_0 \sqsubseteq s_2 \sqsubset s_1)\phi[s_2, s']\}\end{aligned} \tag{3}$$

$$\begin{aligned}&(\exists s')\sigma \sqsubset s' \wedge (\exists s_1 : S_0 \sqsubseteq s_1 \sqsubseteq \sigma)\{\psi[s_1, s'] \wedge \\ &\qquad (\forall s_2 : S_0 \sqsubseteq s_2 \sqsubset s_1)\phi[s_2, s']\}\end{aligned} \tag{4}$$

In the case of the sentence (3), s_1 is a successor situation to σ, putting ψ beyond reach. So we give up checking ψ. Furthermore, for subformula $(\forall s_2 : S_0 \sqsubseteq s_2 \sqsubset s_1)\phi[s_2, s']$, we can only check situations between S_0 and σ; and each time ϕ can be checked "up to" σ only. So with respect to (3) we will require a prefix σ to satisfy the following condition:

$$(\forall s_2 : S_0 \sqsubseteq s_2 \sqsubseteq \sigma)\phi'[s_2, \sigma]$$

In the case of sentence (4) the only limitation is that both ϕ and ψ can be checked only up to σ. The condition derived is this:

$$\begin{aligned}&(\exists s')\sigma \sqsubset s' \wedge \\ &\quad (\exists s_1 : S_0 \sqsubseteq s_1 \sqsubseteq \sigma)\{\psi'[s_1, \sigma] \wedge \\ &\qquad (\forall s_2 : S_0 \sqsubseteq s_2 \sqsubset s_1)\phi'[s_2, \sigma]\}\end{aligned}$$

We have replaced ψ and ϕ with ψ' and ϕ'. The reason is that ψ and ϕ may themselves be temporal expressions. If this is the case, they cannot be verified beyond σ either and so similar conditions as the above two need to be derived for them. In other words, this process must continue recursively on the subformulas of the search control.

The necessary condition we have derived for $\phi \,\mathbf{U}\, \psi$ is the disjunction of the conditions obtained from each case:

$$\begin{aligned}&(\forall s_2 : S_0 \sqsubseteq s_2 \sqsubseteq \sigma)\phi'[s_2, \sigma] \vee \\ &(\exists s_1 : S_0 \sqsubseteq s_1 \sqsubseteq \sigma)\{\psi'[s_1, \sigma] \wedge \\ &\qquad (\forall s_2 : S_0 \sqsubseteq s_2 \sqsubset s_1)\phi'[s_2, \sigma]\}.\end{aligned}$$

In the following definition, we give the condition obtained for each form of search control expression.

Definition 2 *Let ϕ be a search control expression. The prefix control condition induced by ϕ on a sequence s' that starts at s, denoted by $pscc(\phi, s, s')$, is defined as follows:*

1. *if ϕ is a situation suppressed expression, i.e., a search control expression that does not mention any of the temporal operators, then* $pscc(\phi, s, s') = \phi[s]$;
2. $pscc(\circ\phi, s, s') =$ $(\forall a)[do(a, s) \sqsubseteq s' \supset pscc(\phi, do(a, s), s')]$;
3. $pscc(\Diamond\phi, s, s') = True$;
4. $pscc(\Box\phi, s, s') = (\forall s_1 : s \sqsubseteq s_1 \sqsubseteq s')pscc(\phi, s_1, s')$;
5. $pscc(\phi \,\mathbf{U}\, \psi, s, s') =$
$$\begin{aligned}&(\forall s_2 : s \sqsubseteq s_2 \sqsubseteq s')pscc(\phi, s_2, s') \vee \\ &(\exists s_1 : s \sqsubseteq s_1 \sqsubseteq s')\{pscc(\psi, s_1, s') \wedge \\ &\qquad (\forall s_2 : s \sqsubseteq s_2 \sqsubset s_1)pscc(\phi, s_2, s')\};\end{aligned}$$
6. $pscc(\phi \wedge \psi, s, s') = pscc(\phi, s, s') \wedge pscc(\psi, s, s')$;
$pscc(\phi \vee \psi, s', s) = pscc(\phi, s', s) \vee pscc(\psi, s', s)$;
$pscc(\neg\phi, s, s') = \neg pscc(\phi, s, s')$;
$pscc((\exists v)\phi, s, s') = (\exists v)pscc(\phi, s, s')$;
$pscc((\forall v)\phi, s, s') = (\forall v)pscc(\phi, s, s')$.

Example 1 In the briefcase domain, objects can be put in or taken out of a briefcase, and the briefcase can be moved to different locations. When the briefcase is moved to a different location, all objects inside it move to that location as expected. In this domain, $at(x, l, s)$ and $inBcase(x, s)$ are some of the fluents. $at(x, l, s)$ means that object x is at location l in situation s while $inBcase(x, s)$ means that object x is in the briefcase in situation s. A constant Bc represents the briefcase.

One of the search control formulas given in (Bacchus & Kabanza 2000) for the briefcase domain is this:

$$\begin{aligned}&(\forall x, loc)\ \Box\{at(Bc, loc) \wedge goal(at(x, loc)) \supset \\ &\qquad [at(Bc, loc) \,\mathbf{U}\, \neg inBcase(x)]\}\end{aligned}$$

Intuitively, it says that the briefcase must remain in a location until all objects that need to stay in this location according to the goals are taken out of the briefcase.

Denote this control expression by ϕ. Let us find the condition $pscc(\phi, S_0, s)$ (We skip some of the simpler steps).

$$\begin{aligned}&pscc(\phi, S_0, s) = \\ &\qquad (\forall x, loc)(\forall s_1 : S_0 \sqsubseteq s_1 \sqsubseteq s)pscc(\phi_1, s_1, s).\end{aligned}$$

where

$$\phi_1 = \begin{aligned}&at(Bc, loc) \wedge goal(at(x, loc)) \supset \\ &\qquad at(Bc, loc) \,\mathbf{U}\, \neg inBcase(x).\end{aligned}$$

Then,

$$\begin{aligned}&pscc(\phi_1, s_1, s) = \\ &\quad pscc(at(Bc, loc) \wedge goal(at(x, loc)), s_1, s) \supset \\ &\qquad pscc(at(Bc, loc) \,\mathbf{U}\, \neg inBcase(x), s_1, s),\end{aligned}$$

$$\begin{aligned}&pscc(at(Bc, loc) \,\mathbf{U}\, \neg inBcase(x), s_1, s) = \\ &\quad (\forall s_3 : s_1 \sqsubseteq s_3 \sqsubseteq s)pscc(at(Bc, loc), s_3, s) \vee \\ &\quad (\exists s_2 : s_1 \sqsubseteq s_2 \sqsubseteq s)\{pscc(\neg inBcase(x), s_2, s) \wedge \\ &\qquad (\forall s_3 : s_1 \sqsubseteq s_3 \sqsubset s_2)pscc(at(Bc, loc), s_3, s)\},\end{aligned}$$

$$\begin{aligned}&pscc(at(Bc, loc) \wedge goal(at(x, loc)), s_1, s) = \\ &\qquad at(Bc, loc, s_1) \wedge goal(at(x, loc)),\end{aligned}$$

$$pscc(\neg inBcase(x), s_2, s) = \neg inBcase(x, s_2),$$

$$pscc(at(Bc, loc), s_3, s) = at(Bc, loc, s_3).$$

The final condition obtained from ϕ is this:

$$\begin{array}{l}(\forall x, loc)(\forall s_1 : S_0 \sqsubseteq s_1 \sqsubseteq s).\\ at(Bc, loc, s_1) \wedge goal(at(x, loc)) \supset\\ \quad \{(\forall s_3 : s_1 \sqsubseteq s_3 \sqsubseteq s) at(Bc, loc, s_3) \vee\\ \qquad (\exists s_2 : s_1 \sqsubseteq s_2 \sqsubseteq s)[\neg inBcase(x, s_2) \wedge\\ \qquad\quad (\forall s_3 : s_1 \sqsubseteq s_3 \sqsubset s_2) at(Bc, loc, s_3)]\}.\end{array}$$

The following theorem confirms the soundness of the plan prefix conditions in Definition 2 with respect to the original search control expression.

Theorem 1 *Let $\mathcal{D}$ be a basic action theory and ϕ be a search control expression. Then,*

$$\mathcal{D} \models (\forall s).\ (\exists s')\{s \sqsubseteq s' \wedge \phi[S_0, s']\} \supset pscc(\phi, S_0, s).$$

The prefix conditions are certainly not sufficient conditions. It may be impossible to extend a plan prefix that satisfies them into a sequence that satisfies the original control expression. In the next section, we look at the relationship between the above approach to enforcing control and the use of control through the progression algorithm, and show that the conditions eliminate the same prefixes progression does.

Progression

Bacchus & Kabanza's progression algorithm takes as input a linear temporal formula τ and a world w, and computes a temporal formula τ'. Intuitively, τ' is the formula that needs to be checked in the successor world of w. The same algorithm can be used to compute the progression of a search control formula but with respect to a situation s instead of a world.

We will define an operator $Pr(\phi, s)$ for computing the progression of an expression ϕ in a situation s. The control operand ϕ is a hybrid expression: it may have both situation suppressed expressions and situation calculus formulas as subexpressions. We will refer to this type of expressions as *h-expressions*. The output of $Pr(\phi, s)$ is also an h-expression.

Definition 3 *Let ϕ be an h-expression. The progression of ϕ in s, $Pr(\phi, s)$, is defined as follows:*

1. *if ϕ is an atomic situation suppressed expression, then $Pr(\phi, s) = \phi[s]$;*
2. *if ϕ is a situation calculus atom, then $Pr(\phi, s) = \phi$;*
3. *$Pr(\circ\phi, s) = \phi$;*
4. *$Pr(\phi\ \mathbf{U}\ \psi, s) = Pr(\psi, s) \vee (Pr(\phi, s) \wedge \phi\ \mathbf{U}\ \psi)$;*
5. *$Pr(\Diamond\phi, s) = Pr(\phi, s) \vee \Diamond\phi$;*
6. *$Pr(\Box\phi, s) = Pr(\phi, s) \wedge \Box\phi$;*
7. *$Pr(\phi \wedge \psi, s) = Pr(\phi, s) \wedge Pr(\psi, s)$;*
8. *$Pr(\phi \vee \psi, s) = Pr(\phi, s) \vee Pr(\psi, s)$;*
9. *$Pr(\neg\phi, s) = \neg Pr(\phi, s)$;*
10. *$Pr((\forall v)\phi, s) = (\forall v) Pr(\phi, s)$;*
11. *$Pr((\exists v)\phi, s) = (\exists v) Pr(\phi, s)$.*

The progression algorithm as defined in (Bacchus & Kabanza 2000) transforms universal and existential quantification into a conjunction and a disjunction, respectively. Since such a transformation does not produce an equivalent formula, the algorithm requires the use of "bounded quantification" instead of normal logical quantification. For the purpose of analyzing the relationship between our plan prefix conditions and the progression algorithm, the restriction of bounded quantification is unnecessary. Progression of quantified formulas can be handled in the simple manner shown above.

The Bacchus-Kabanza progression algorithm computes a temporal condition the next world must satisfy. Similarly, the operator $Pr(\phi, s)$ computes the control condition that is to be checked in the next situation. In terms of sequences of actions, it computes the condition that needs to be checked on one-action sequence. For our analysis, we need to define an operator that computes the condition for a sequence of any length. This can be defined by recursion:

Definition 4 *The progression of a control expression ϕ through a sequence $do(\vec{\alpha}, s)$, denoted by $Pr^*(\phi, do(\vec{\alpha}, s))$, is defined as follows:*

$$\begin{array}{l}Pr^*(\phi, s) = Pr(\phi, s)\\ Pr^*(\phi, do(\alpha, \sigma)) = Pr(Pr^*(\phi, \sigma), do(\alpha, \sigma)).\end{array}$$

For example, for the control $\Box \circ Q$, where Q is a fluent, and a sequence $do([A_1, A_2, A_3], S_0)$ we have:

$$\begin{array}{l}Pr^*(\Box \circ Q, do([A_1, A_2, A_3], S_0)) =\\ \quad Q(do(A_1, S_0)) \wedge\\ \quad Q(do([A_1, A_2], S_0)) \wedge\\ \quad Q(do([A_1, A_2, A_3], S_0)) \wedge\\ \quad Q \wedge \Box \circ Q\end{array}$$

As the above example shows, the result of progression through a sequence may contain subexpressions that are not situation calculus formulas. These expressions are parts of the control that must hold in future situations, but say nothing about the current plan prefix. Thus, when checking the control on a prefix, we will ignore them by assuming they are satisfied: for an h-expression ϕ, let $\|\phi\|$ denote the situation calculus formula obtained from ϕ by replacing every control subexpression with $True$.

The following theorem tells us that the prefix control conditions from Definition 2 are exactly what the progression algorithm checks on partial plans.

Theorem 2 Let $do(\vec{\alpha}, s)$ be a situation term, s be a situation variable or S_0, and ϕ be a search control expression. Then,

$$\mathcal{D} \models (\forall).\{\|Pr^*(\phi, do(\vec{\alpha}, s))\| \equiv pscc(\phi, s, do(\vec{\alpha}, s))\}.$$

Proof: (sketch) We begin with a lemma that Pr^* distributes over the propositional connectives and over quantifiers:

Lemma 1 *For all $\phi, \psi, \sigma', \sigma$,*

1. *$Pr^*(\phi \wedge \psi, \sigma', \sigma) = Pr^*(\phi, \sigma', \sigma) \wedge Pr^*(\phi, \sigma', \sigma)$;*
2. *$Pr^*(\phi \vee \psi, \sigma', \sigma) = Pr^*(\phi, \sigma', \sigma) \vee Pr^*(\phi, \sigma', \sigma)$;*

3. $Pr^*(\neg\phi, \sigma', \sigma) = \neg Pr^*(\phi, \sigma', \sigma)$;
4. $Pr^*((\exists x)\phi, \sigma', \sigma) = (\exists x)Pr^*(\phi, \sigma', \sigma)$;
5. $Pr^*((\forall x)\phi, \sigma', \sigma) = (\forall x)Pr^*(\phi, \sigma', \sigma)$.

The proof of this lemma is straight forward from the definitions of Pr and Pr^*.

The following equivalent but less elegant recursive version of Pr^* allows a simpler proof and so we will use it:

$$\begin{array}{l} Pr'(\phi, s, s) = Pr(\phi, s) \\ Pr'(\phi, s, do(\vec{\alpha}_k, s)) = Pr'(Pr(\phi, s), do(\alpha_1, s), do(\vec{\alpha}_k, s)). \end{array}$$

Furthermore, we prove a slightly stronger theorem: let $\vec{\alpha}$ be a sequence of action terms $\alpha_1, \ldots, \alpha_k$, $do(\vec{\alpha}, s)$ be a situation term, and ϕ be an h-expression. We prove: For every prefix $\sigma = do([\alpha_1, \ldots, \alpha_i], s)$ of $do(\vec{\alpha}, s)$

$$\mathcal{D} \models (\forall).\|Pr'(\phi, \sigma, do(\vec{\alpha}, s))\| \equiv pscc(\phi, \sigma, do(\vec{\alpha}, s)).$$

The proof is by induction on the structure of ϕ and the difference between the length of $do(\vec{\alpha}, s)$ and σ.

Assume the theorem on h-expressions ϕ and ψ. Let us prove the theorem for the expression $\phi \,\mathbf{U}\, \psi$.

For the base case, we have that

$$\begin{array}{ll} Pr'(\phi \,\mathbf{U}\, \psi, s, s) & = Pr(\phi \,\mathbf{U}\, \psi, s) \\ & = Pr(\psi, s) \vee (Pr(\phi, s) \wedge \phi \,\mathbf{U}\, \psi). \end{array}$$

Hence,

$$\begin{array}{l} \|Pr'(\phi \,\mathbf{U}\, \psi, s, s)\| = \\ \qquad \|Pr(\psi, s)\| \vee (\|Pr(\phi, s)\| \wedge \|\phi \,\mathbf{U}\, \psi\|) \\ \qquad = \|Pr(\psi, s)\| \vee (\|Pr(\phi, s)\| \wedge True) \\ \qquad \equiv \|Pr(\psi, s)\| \vee \|Pr(\phi, s)\|. \end{array}$$

On the other hand we have:

$$\begin{array}{l} pscc(\phi \,\mathbf{U}\, \psi, s, s) = \\ (\forall s_2 : s \sqsubseteq s_2 \sqsubseteq s) pscc(\phi, s_2, s) \vee \\ (\exists s_1 : s \sqsubseteq s_1 \sqsubseteq s)\{pscc(\psi, s_1, s) \wedge \\ \qquad (\forall s_2 : s \sqsubseteq s_2 \sqsubset s_1) pscc(\phi, s_2, s)\} \\ \equiv \\ pscc(\phi, s, s) \vee pscc(\psi, s, s). \end{array}$$

The theorem follows from this and the induction assumption on ϕ, ψ.

Now, consider a prefix σ' of $\sigma = do([\alpha_1, \ldots, \alpha_k], s)$ such that the length, l, of σ' is strictly less than k. That is, σ' consists of the first l action terms in σ.

We have:

$$\begin{array}{l} Pr'(\phi \,\mathbf{U}\, \psi, \sigma', \sigma) \\ = Pr'(\ Pr(\phi \,\mathbf{U}\, \psi, \sigma'),\ do(\alpha_{l+1}, \sigma'), \sigma) \\ = Pr'(\ Pr(\psi, \sigma') \vee (Pr(\phi, \sigma') \wedge \phi \,\mathbf{U}\, \psi),\ do(\alpha_{l+1}, \sigma'), \sigma) \\ = Pr'(\ Pr(\psi, \sigma'),\ do(\alpha_{l+1}, \sigma'), \sigma) \vee \\ \qquad Pr'(\ Pr(\phi, \sigma'),\ do(\alpha_{l+1}, \sigma'), \sigma) \wedge \\ \qquad Pr'(\ \phi \,\mathbf{U}\, \psi,\ do(\alpha_{l+1}, \sigma'), \sigma). \end{array}$$

Thus, by definition of Pr', we have that

$$\begin{array}{l} Pr'(\phi \,\mathbf{U}\, \psi, \sigma', \sigma) = \\ \qquad Pr'(\psi, \sigma', \sigma) \vee \\ \qquad Pr'(\phi, \sigma', \sigma) \wedge Pr'(\phi \,\mathbf{U}\, \psi, do(\alpha_{l+1}, \sigma'), \sigma). \end{array} \tag{5}$$

On the other hand we have that

$$\begin{array}{l} pscc(\phi \,\mathbf{U}\, \psi, \sigma', \sigma) = \\ (\forall s_2 : \sigma' \sqsubseteq s_2 \sqsubseteq \sigma) pscc(\phi, s_2, \sigma) \vee \\ (\exists s_1 : \sigma' \sqsubseteq s_1 \sqsubseteq \sigma)\{pscc(\psi, s_1, \sigma) \wedge \\ \qquad (\forall s_2 : \sigma' \sqsubseteq s_2 \sqsubset s_1) pscc(\phi, s_2, \sigma)\} \end{array}$$

$$\begin{array}{l} \equiv \\ pscc(\phi, \sigma', \sigma) \wedge \\ (\forall s_2 : do(\alpha_{l+1}, \sigma') \sqsubseteq s_2 \sqsubseteq \sigma) pscc(\phi, s_2, \sigma) \\ \vee \\ pscc(\psi, \sigma', \sigma) \wedge \\ (\forall s_2 : \sigma' \sqsubseteq s_2 \sqsubset \sigma') pscc(\phi, s_2, \sigma) \\ \vee \\ (\exists s_1 : do(\alpha_{l+1}, \sigma') \sqsubseteq s_1 \sqsubseteq \sigma)\{pscc(\psi, s_1, \sigma) \wedge \\ (\forall s_2 : do(\alpha_{l+1}, \sigma') \sqsubseteq s_2 \sqsubset s_1) pscc(\phi, s_2, \sigma) \wedge \\ \qquad\qquad pscc(\phi, \sigma', \sigma)\} \end{array}$$

$$\begin{array}{l} \equiv \\ pscc(\phi, \sigma', \sigma) \wedge \\ (\forall s_2 : do(\alpha_{l+1}, \sigma') \sqsubseteq s_2 \sqsubseteq \sigma) pscc(\phi, s_2, \sigma) \\ \vee \\ pscc(\psi, \sigma', \sigma) \\ \vee \\ pscc(\phi, \sigma', \sigma) \wedge \\ (\exists s_1 : do(\alpha_{l+1}, \sigma') \sqsubseteq s_1 \sqsubseteq \sigma)\{pscc(\psi, s_1, \sigma) \wedge \\ \qquad (\forall s_2 : do(\alpha_{l+1}, \sigma') \sqsubseteq s_2 \sqsubset s_1) pscc(\phi, s_2, \sigma)\} \end{array}$$

$$\begin{array}{l} \equiv \\ pscc(\psi, \sigma', \sigma) \vee \\ \{pscc(\phi, \sigma', \sigma) \wedge \\ \quad (\forall s_2 : do(\alpha_{l+1}, \sigma') \sqsubseteq s_2 \sqsubseteq \sigma) pscc(\phi, s_2, \sigma) \vee \\ \quad (\exists s_1 : do(\alpha_{l+1}, \sigma') \sqsubseteq s_1 \sqsubseteq \sigma)\{pscc(\psi, s_1, \sigma) \wedge \\ \qquad (\forall s_2 : do(\alpha_{l+1}, \sigma') \sqsubseteq s_2 \sqsubset s_1) pscc(\phi, s_2, \sigma)\}\} \end{array}$$

$$\begin{array}{l} \equiv \\ pscc(\psi, \sigma', \sigma) \vee \\ (pscc(\phi, \sigma', \sigma) \wedge pscc(\phi \,\mathbf{U}\, \psi, do(\alpha_{l+1}, \sigma'), \sigma)) \end{array}$$

The theorem then follows from this, (5), and the induction assumptions.

The proof of the theorem for other forms of control expressions is similar. □

Control Conditions in the Action Theory

The plan prefix conditions denoted by $pscc(\phi, s', s)$ are situation calculus *bounded formulas*, a class of formulas that can be used as action preconditions in nonMarkovian action theories (Gabaldon 2002), and in some cases they can be compiled into preconditions, as shown in (Gabaldon 2003), of the classical Markovian form.

Control into nonMarkovian Theories

The simplest way to incorporate the plan prefix conditions into an action theory is by allowing nonMarkovian axioms. In this case we can simply add $pscc(\phi, S_0, s)$ as an additional precondition to every action precondition axiom as follows:

$$\begin{array}{c} Poss(A(\vec{x}), s) \equiv \\ \Pi_A(\vec{x}, s) \wedge pscc(\phi, S_0, do(A(\vec{x}), s)). \end{array} \tag{6}$$

Intuitively, we are treating actions that violate the control conditions as if they were impossible. Since a planner only adds to the current sequence actions that are possible, actions that violate the control are thus never added to a plan prefix.

In this setting, determining whether an action is executable in a given situation amounts to solving a *projection problem*. Following (Reiter 1991; 2001), we solve such a problem in the situation calculus by applying *regression*. In previous work (Gabaldon 2002) we generalized Reiter's regression operator, $\mathcal{R}$, so that it can be applied on formulas such as our control conditions above. In other words, we extended the class of regressable formulas defined by Reiter with formulas that, under some restrictions, mention $\sqsubset$ and quantify over situations. This allows us to solve projection problems with respect to nonMarkovian action theories.

At first thought it may seem counterintuitive to use regression with forward-chaining planning. But forward-chaining planning is used for generating candidate sequences of actions, i.e., plan prefixes, while regression is used for solving projection problems once we have a candidate sequence in hand. In other words, forward-chaining is used for generating a sequence and regression for the independent problem of testing properties against it.

Control into Markovian Theories by Compiling

The other alternative for incorporating the control conditions, which seems to be computationally advantageous, is to apply the transformation operator $\mathcal{M}$ from (Gabaldon 2003) to the theory obtained as described in the previous subsection and "Markovianize" it. This is achieved by replacing each action precondition axiom (6) with the following:

$$Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s) \wedge \mathcal{M}[pscc(\phi, S_0, do(A(\vec{x}), s))].$$

The transformation operator $\mathcal{M}$ is roughly defined as follows:

- Suppose W is a formula whose only variable of sort situation is s, then $\mathcal{M}[W] = \mathcal{R}^s[W]$.

  Here, $\mathcal{R}^s[W]$ is the result of applying regression steps until all situation terms are reduced to variable s, that is, until all terms $do(\vec{\alpha}, s)$ have been regressed down to s.

- Suppose W is a formula of the form

  $$(\exists s_1 : S_0 \sqsubseteq s_1 \sqsubseteq do(\vec{\alpha}, s))W'$$

  where W' is bounded by s_1. Intuitively, if a formula is bounded by s, then all variables of sort situation that appear in it are restricted to range over predecessors of situation terms of the form $do(\vec{\alpha}, s)$, i.e., over past situations relative to the current situation s or relative to a future situation $do(\vec{\alpha}, s)$.

  Then,

  $$\mathcal{M}[W] = \mathcal{M}[P_W(\vec{x}, do(\vec{\alpha}, s))]$$

  where $\vec{x}$ are all the free non-situation variables in the original formula W. P_W is a new fluent that is added to the theory. It has the following successor state axiom:

  $$P_W(\vec{x}, do(a, s)) \equiv \mathcal{M}[W'|^{s_1}_{do(a,s)}] \vee P_W(\vec{x}, s).$$

  Here, $W|^s_\sigma$ denotes the formula obtained from W by replacing the variable s everywhere in W with the term σ. The new fluent P_W is initially set to false: $\neg P_W(\vec{x}, S_0)$.

- Suppose W is a formula of the form:

  $$\begin{array}{l} (\exists s_1 : S_0 \sqsubseteq s_1 \sqsubseteq do(\vec{\alpha}, s))\{W_1 \wedge \\ \qquad (\forall s_2 : s_1 \sqsubset s_2 \sqsubset do(\vec{\alpha}, s))W_2\} \end{array}$$

  where W_1, W_2 are bounded by s_1, s_2 respectively. Then,

  $$\mathcal{M}[W] = \mathcal{M}[P_W(\vec{x}, do(\vec{\alpha}, s))]$$

  where $\vec{x}$ are all the free non-situation variables in W and P_W is a new fluent with successor state axiom:

  $$\begin{array}{c} P_W(\vec{x}, do(a, s)) \equiv \\ \mathcal{M}[W_1|^{s_1}_{do(a,s)}] \vee \{\mathcal{M}[W_2|^{s_2}_{s}] \ \wedge \ P_W(\vec{x}, s)\} \end{array}$$

  and initially false: $\neg P_W(\vec{x}, S_0)$.

- The remaining cases are simply:

  $$\begin{array}{l} \mathcal{M}[\neg W] = \neg \mathcal{M}[W], \\ \mathcal{M}[W_1 \wedge W_2] = \mathcal{M}[W_1] \wedge \mathcal{M}[W_2], \\ \mathcal{M}[(\exists v)W] = (\exists v)\mathcal{M}[W]. \end{array}$$

One advantage is that the resulting axioms only refer to the current situation s. Another advantage of applying the transformation is that when this is done, the control precondition in each axiom becomes specific to its action type[3] $A(\vec{x})$, due to the fact that $do(A(\vec{x}), s))$ is an argument of $pscc$. Since the transformation works in part by applying regression steps, the result is simpler, action type specific control conditions. The following example illustrates this.

Example 2 Consider the following briefcase domain control expression

$$at(Bc, L) \ \mathbf{U} \ \neg inBcase(A) \tag{7}$$

saying that the briefcase Bc must stay at the location L until the object A is taken out of it (say because one of the goals is that object A be at location L).

[3]There can be an infinite number of action *instances* even if there is a finite number of action *types*, i.e., action functions, in the language.

Suppose that this domain includes the action $move(x, loc)$, for moving an object x to a location loc, which is possible whenever the target location loc is different than the current location of the object x. Thus we would have the following precondition axiom for this action:

$$Poss(move(x, loc), s) \equiv \\ (\exists cloc).at(x, cloc, s) \wedge cloc \neq loc.$$

For the sake of simplicity, let us assume that this is the only action that affects the at, so the dynamics of this fluent are captured by the following successor state axiom which states that an object is at a location loc iff it has just been move there or it was already there and it has not just been moved anywhere:

$$at(x, loc, do(a, s)) \equiv a = move(x, loc) \vee \\ at(x, loc, s) \wedge \neg(\exists l)a = move(x, l).$$

We will also need the successor state axiom of the fluent $inBc(x, s)$ which means that object x is inside the briefcase in situation s:

$$inBc(x, do(a, s)) \equiv a = putIntoBc(x) \vee \\ inBc(x, s) \wedge a \neq takeOutofBc(x).$$

The prefix condition obtained from the control expression (7) is the following:

$$pscc((7),\ S_0, do(move(x, loc), s)) = \\ (\forall s_2\colon S_0 \sqsubseteq s_2 \sqsubseteq do(move(x, loc), s))at(Bc, L, s_2) \vee \\ (\exists s_1\colon S_0 \sqsubseteq s_1 \sqsubseteq do(move(x, loc), s)) \\ [\neg inBcase(A, s_1) \wedge (\forall s_2\colon S_0 \sqsubseteq s_2 \sqsubset s_1)at(Bc, L, s_2)].$$

Let us apply transformation $\mathcal{M}$ to this formula. Consider the first disjunct, put in the equivalent form:

$$\neg(\exists s_2\colon S_0 \sqsubseteq s_2 \sqsubseteq do(move(x, loc), s))\neg at(Bc, L, s_2)$$

From this, the transformation results in

$$\mathcal{M}[\neg P_1(do(move(x, loc), s))] = \\ \mathcal{R}^s[\neg P_1(do(move(x, loc), s))].$$

The new fluent P_1 has the following successor state axiom:

$$P_1(do(a, s)) \equiv \\ \mathcal{M}[\neg at(Bc, L, do(a, s)) \vee P_1(s) \\ \equiv \mathcal{R}^s[\neg at(Bc, L, do(a, s)) \vee P_1(s) \\ \equiv a \neq move(Bc, L) \wedge \\ [\neg at(Bc, L, s) \vee (\exists l)a = move(Bc, l)]\ \vee P_1(s).$$

Hence $\mathcal{R}^s[\neg P_1(do(move(x, loc), s))]$ is equal to:

$$\neg\{move(x, loc) \neq move(Bc, L) \wedge \\ [\neg at(Bc, L, s) \vee (\exists l)move(x, loc) = move(Bc, l)] \vee \\ P_1(s)\} \\ \equiv \\ x = Bc \wedge loc = L \wedge \neg P_1(s).$$

From the second disjunct, we obtain a similar $\mathcal{M}[P_2(do(move(x, loc), s))]$ and a new fluent P_2 with the following successor state axiom:

$$P_2(do(a, s)) \equiv \\ \mathcal{R}^s[\neg inBc(A, do(a, s))] \wedge \\ \mathcal{M}[(\forall s_2\colon S_0 \sqsubseteq s_2 \sqsubset do(a, s))at(Bc, L, s_2)] \vee \\ P_2(s) \\ = \\ \mathcal{R}^s[\neg inBc(A, do(a, s))]\ \wedge \mathcal{M}[\neg P_1(do(a, s))]\ \vee P_2(s) \\ \equiv \\ a \neq putIntoBc(A) \wedge \\ [\neg inBc(A, s) \vee a = takeOutofBc(A)] \wedge \\ \{a = move(Bc, L) \vee \\ at(Bc, L, s) \wedge \neg(\exists l)a = move(Bc, l)\}\ \wedge \neg P_1(s) \vee \\ P_2(s).$$

And so we have that $\mathcal{M}[P_2(do(move(x, loc), s))]$ is equal to $\mathcal{R}^s[\neg P_2(do(move(x, loc), s))]$, which in turn is equal to:

$$move(x, loc) \neq putIntoBc(A) \wedge \\ [\neg inBc(A, s) \vee move(x, loc) = takeOutofBc(A)] \wedge \\ \{move(x, loc) = move(Bc, L) \vee at(Bc, L, s) \wedge \\ \neg(\exists l)move(x, loc) = move(Bc, l)\} \wedge \\ \neg P_1(s) \vee \\ P_2(s) \\ \equiv \\ \neg inBc(A, s) \wedge x = Bc \wedge loc = L \wedge \neg P_1(s) \vee \\ P_2(s).$$

We thus obtain, by applying the transformation, the compiled prefix condition:

$$x = Bc \wedge loc = L \wedge \neg P_1(s) \vee \\ \neg inBc(A, s) \wedge x = Bc \wedge loc = L \wedge \neg P_1(s)\ \vee P_2(s) \\ \equiv \\ x = Bc \wedge loc = L \wedge \neg P_1(s)\ \vee P_2(s).$$

Applying the transformation $\mathcal{M}$ yields the following precondition axiom for $move(x, loc)$:

$$Poss(move(x, loc), s) \equiv \\ (\exists cloc).at(x, cloc, s) \wedge cloc \neq loc \wedge \\ x = Bc \wedge loc = L \wedge \neg P_1(s)\ \vee P_2(s).$$

Related Work

As we have mentioned above, the approach to planning with search control we have considered in this paper was introduced by (Bacchus & Kabanza 2000). The TALPlanner system of (Kvarnström & Doherty 2000) follows Bacchus & Kabanza's approach: it utilizes search control knowledge expressed first-order linear temporal logic and Bacchus& Kabanza's progression algorithm. It also provides the alternative to express the control knowledge in the system's underlying logical language TAL. The TALPlanner system has been extended to incorporate some forms of prefix condition extraction that is similar to what we have considered here. (Kvarnström 2002) shows how to extract prefix conditions from some types of search control formulas in the TAL language The class of control formulas he considers corresponds to formulas of the form $\Box\phi$ where ϕ is a formula that mentions no temporal operator other than $\circ$ (next), and this cannot be nested. In this paper we show how to obtain prefix conditions from general temporal logic formulas for nonMarkovian basic action theories. Some of these conditions can be compiled to produce a Markovian theory. The conditions we obtain from the aforementioned $\Box\phi$ formulas are indeed among those that can be compiled, but the class is larger. Example 2 shows an example with a formula using the operator $\mathbf{U}$ (until). Example 1 shows a formula of the form $\Box[P \supset (Q\ \mathbf{U}\ R)]$, where P, Q, R are non-temporal. This is a very useful form of control formula that also yields compilable prefix conditions in our framework. On the other hand the TALPlanner system utilizes several other formula optimization techniques that we have not considered. We are currently working on generalizing the transformation operator so that it can be used to compile all control expressions ϕ of Definition 1.

Reiter (2001) also discusses the use of domain dependent control knowledge with several simple planners written in the high level, situation calculus based language Golog. Reiter encodes control knowledge in the definition of a predicate $badSituation(s)$. Reiter's programs are also forward-chaining planners and intuitively work as follows: an action a is chosen, then it is executed by the program (thereby checking that the action's preconditions hold) and then the test $\neg badSituation?$ is executed. This test amounts to checking if $\neg badSituation(do(a, s))$ holds, where s is the current plan prefix. The planner proceeds recursively until a situation that satisfies the goal is found or a bound is reached. By using the $badSituation$ predicate instead of using the search control as further preconditions, Reiter emphasizes that control constraints are not executability conditions although both have the effect of eliminating sequences of actions. We could easily adapt our approach and use our plan prefix conditions to define bad situations instead of adding them as further preconditions. As presented in his book, Reiter's predicate $badSituation$ must be defined by an axiom:

$$badSituation(s) \equiv W$$

where W is a formula whose only situation variable is s. The predecessor relation $\sqsubset$ on situations cannot appear in W nor any situation variable other than s. However, using the approach we have presented above, we could take a control expression ϕ and define $badSituation$ as follows:

$$badSituation(s) \equiv \neg\mathcal{M}[pscc(\phi, S_0, s)].$$

That is, a situation is bad if it falsifies the prefix control conditions. We can then run Reiter's Golog planning programs without modification.

Also using the situation calculus as a formal framework, Lin has considered search control knowledge in planning. In (Lin 1997), he considers control knowledge in the form of a *subgoal ordering*: given a conjunctive goal $g_1\& \ldots \& g_k$, the control knowledge is expressed in the form of precedence relation statements of the form $g_i \prec g_j$ saying that any plan that achieves G, achieves the goal g_i first and later achieves g_j while having maintained g_i. Lin shows how subgoal ordering knowledge can be derived from the goal and the background theory in the situation calculus and how it can be employed in planning algorithms. In (Lin 1998), he formalizes control knowledge that is in the form of a partial order on actions, in order to provide a situation calculus semantics to the Prolog cut operator. Our work is in the spirit of Lin's, of "the situation calculus as a general framework for representing and reasoning about control and strategic information in problem solving."

Conclusion

We have considered the problem of planning with declarative search control of the form used in Bacchus & Kabanza's TLPlan system, using the situation calculus as a formal framework. We start with Green's specification of the planning problem and modify it by adding search control to the problem. In the modified specification, search control is viewed as a property of a sequence that possibly extends a plan. Then, using this specification as a guide, we derive some conditions that must be satisfied by every plan prefix. These conditions are situation calculus formulas of a form on which regression can be used. We then show that these conditions correspond exactly to the temporal progression algorithm used in TLPlan, i.e., a prefix is eliminated through these conditions iff the progression algorithm eliminates it. Finally, we show how the control conditions can be directly incorporated into the precondition axioms of a nonMarkovian action theory or, through a transformation, into a classic Markovian one.

Acknowledgments I am very thankful to Fahiem Bacchus and Yves Lespérance for useful discussions on the subject of this paper. Also my thanks to the reviewers for their useful comments.

References

Bacchus, F., and Ady, M. 1999. Precondition control. Available at http://cs.toronto.edu/∼fbacchus/on-line.html.

Bacchus, F., and Kabanza, F. 1998. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence* 22:5–27.

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116:123–191.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1996. Complexity results for hierarchical task-network planning. *Annals of Mathematics and Artificial Intelligence* 18:69–93.

Gabaldon, A. 2002. Non-markovian control in the situation calculus. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI'02)*, 519–524.

Gabaldon, A. 2003. Compiling control knowledge into preconditions for planning in the situation calculus. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03).*

Green, C. 1969. Theorem proving by resolution as a basis for question-answering systems. In Meltzer, B., and Michie, D., eds., *Machine Intelligence 4.* New York: American Elsevier. 183–205.

Kautz, H., and Selman, B. 1998. The role of domain-specific knowledge in the planning as satisfiability framework. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS'98)*, 181–189.

Kvarnström, J., and Doherty, P. 2000. TALplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence* 30:119–169.

Kvarnström, J. 2002. Applying domain analysis techniques for domain-dependent control in TALPlanner. In *Procs. of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS'02)*, 369–378.

Lin, F. 1997. An ordering of subgoals for planning. *Annals of Mathematics and Artificial Intelligence. (Special issue in honor of Professor Michael Gelfond)* 31(1–3):59–83.

Lin, F. 1998. Applications of the situation calculus to formalizing control and strategic information: the Prolog cut operator. *Artificial Intelligence* 103(1–2):273–294.

McCarthy, J. 1963. Situations, actions and causal laws. Technical report, Stanford University. Reprinted in Semantic Information Processing (M. Minsky ed.), MIT Press, Cambridge, Mass., 1968, pp. 410–417.

Pirri, F., and Reiter, R. 1999. Some contributions to the metatheory of the Situation Calculus. *Journal of the ACM* 46(3):325–364.

Reiter, R. 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V., ed., *Artificial Intelligence and Mathematical Theory of Computation*. Academic Press. 359–380.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems*. Cambridge, MA: MIT Press.

Rintanen, J. 2000. Incorporation of temporal logic control into plan operators. In Horn, W., ed., *Procs. of the 14th European Conference on Artificial Intelligence (ECAI'00)*, 526–530. Amsterdam: IOS Press.

Sacerdoti, E. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* 5:115–135.

Wilkins, D. E. 1988. *Practical Planning: Extending the classic AI planning paradigm*. San Mateo, CA: Morgan Kaufmann.