When is Planning Decidable?*

Kutluhan Erol kutluhan@cs.umd.edu
Dana S. Nau† nau@cs.umd.edu
V. S. Subrahmanian‡ vs@cs.umd.edu

Computer Science Department
University of Maryland
College Park, MD 20742

Abstract

In this paper, we show conditions under which planning is decidable and undecidable. Our results on this topic solve an open problem posed in (Chapman 1987), and clear up some difficulties with his undecidability theorems.

1 Introduction

Much planning research has been motivated, in one way or another, by the difficulty of producing complete and correct plans. For example, abstraction (Sacerdoti 1974, Charniak & McDermott 1985, Nilsson 1980, Tate *et al.* 1990) and task reduction (Sacerdoti 1975, Charniak & McDermott 1985, Tate *et al.* 1990) were developed in an effort to make planning more efficient, and concepts such as deleted-condition interactions were developed to describe situations which make planning difficult.

Here we examine how the decidability of domain-independent planning depends on the nature of the planning operators. We consider planning problems in which the current state is a set of ground atoms, and each planning operator is a STRIPS-style operator consisting of three lists of atoms: a precondition list, an add list, and a delete list. Our results can be summarized as follows:

1. If function symbols are allowed or if the language contains infinitely many constant symbols, then determining, in general, whether a plan exists is *undecidable* (more specifically, semidecidable)[1]. This is true even if we have no delete lists and the precondition list of each operator contains at most one (non-negated) atom. If no function symbols are allowed and only finitely many constant symbols are allowed, then plan existence is *decidable*, regardless of the presence or absence of delete lists.
2. The above results resolve an open problem stated by Chapman in (Chapman 1987): whether or not planning is undecidable if the initial state is finite. If the initial state is finite and the language has finitely many ground terms, then the general planning problem is decidable, but if the initial state is finite and the language has infinitely many ground terms, then planning is undecidable in general.
3. Chapman's Second Undecidability Theorem states that "planning is undecidable even with a finite initial situation if the action representation is extended to represent actions whose effects are a function of their situation" (Chapman 1987). This theorem, as stated, is misleading. Whether or not planning is undecidable has nothing to do with whether or not the operators have conditional effects—instead, planning is decidable if and only if the language contains finitely many ground terms.

Section 2 contains the basic definitions. Section 3 contains the decidability and undecidability results, and Section 4 compares and contrasts them with Chapman's results. Section 5 contains concluding remarks. For proofs of the theorems, the reader is referred to (Erol *et al.* 1991).

2 Preliminaries

Throughout this section, we let $\mathcal{L}$ be an ordinary first-order language generated by finitely many constant,

*This work was supported in part by NSF Grant NSFD CDR-88003012 to the University of Maryland Systems Research Center, as well as NSF grants IRI-8907890 and IRI-9109755.

†Also in the Systems Research Center and the Institute for Advanced Computer Studies.

‡Also in the Institute for Advanced Computer Studies

[1]The formal definition of the plan existence problem is given at the end of the section entitled "Preliminaries."

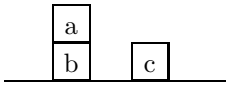

Fig. 1: Initial configuration

Fig. 2: Goal configuration

function and predicate symbols. $\mathcal{L}$ has an infinite number of variable symbols together with the usual logical symbols.

A *state* is a set of ground atoms. Intuitively, a state tells us which ground atoms are currently true. Thus, if a ground atom A is in state S, then A is true in state S; if $B \notin S$, then B is false in state S. Thus, a state is simply an Herbrand interpretation for the language $\mathcal{L}$, and hence each formula of first-order logic is either satisfied or not satisfied in S according to the usual first-order logic definition of satisfaction.

We consider a STRIPS-style *planning operator* α to be a 4-tuple $(\text{Name}(\alpha), \text{Pre}(\alpha), \text{Add}(\alpha), \text{Del}(\alpha))$, where

1. $\text{Name}(\alpha)$ is a syntactic expression of the form $\alpha(X_1, \ldots, X_n)$ where each X_i is a variable symbol of $\mathcal{L}$;
2. $\text{Pre}(\alpha)$ is a finite set of literals, called the *precondition list* of α, whose variables are all from the set $\{X_1, \ldots, X_n\}$;
3. $\text{Add}(\alpha)$ and $\text{Del}(\alpha)$ are both finite sets of atoms (possibly non-ground) whose variables are taken from the set $\{X_1, \ldots, X_n\}$. $\text{Add}(\alpha)$ is called the *add list* of α, and $\text{Del}(\alpha)$ is called the *delete list* of α.

Observe that atoms and negated atoms may occur in the precondition list, but negated atoms may not occur in either the add list or the delete list.

A *planning domain* is a pair $\mathbf{P} = (S_0, \mathcal{O})$, where S_0 is the *initial state* and $\mathcal{O}$ is a finite set of operators. A *goal* is an existentially closed conjunction of atoms. A *planning problem* is a triple $\mathbf{P} = (S_0, \mathcal{O}, G)$, where $(S_0, \mathcal{O})$ is a planning domain and G is a goal.

Example 1 (Blocks World) Consider a blocks-world domain containing three blocks a, b, c, along with Nilsson's "stack", "unstack", "pickup", and "putdown" operators (Nilsson 1980). Suppose the initial and goal configurations are as shown in Figs. 1 and 2. This domain can be represented as follows:

- *Language.* The language $\mathcal{L}$ contains a supply of variable symbols $X_1, X_2, \ldots$, and three constant symbols a, b, c to represent the three blocks. $\mathcal{L}$ contains a binary predicate symbol "on", unary predicate symbols "ontable", "clear", and "holding", and a 0-ary predicate symbol "handempty". Operator names, such as "stack", "unstack", etc., are not part of $\mathcal{L}$.
- *Operators.* The "unstack" operator is the following 4-tuple (the "stack", "pickup", and "putdown" operators are defined analogously):

$$\begin{aligned} \text{Name}: &\quad \text{unstack}(X_1, X_2) \\ \text{Pre} &\quad \{\text{on}(X_1, X_2), \text{clear}(X_1), \text{handempty}()\} \\ \text{Del}: &\quad \{\text{on}(X_1, X_2), \text{clear}(X_1), \text{handempty}()\} \\ \text{Add}: &\quad \{\text{clear}(X_2), \text{holding}(X_1)\} \end{aligned}$$

- *Planning Domain.* The planning domain is $(S_0, \mathcal{O})$, where S_0 and $\mathcal{O}$ are as follows:

$$\begin{aligned} S_0 &= \{\text{clear}(a), \text{on}(a, b), \text{ontable}(b), \\ &\quad \text{clear}(c), \text{ontable}(c), \text{handempty}()\}; \\ \mathcal{O} &= \{\text{stack}, \text{unstack}, \text{pickup}, \text{putdown}\}. \end{aligned}$$

- *Planning Problem.* The planning problem is $(S_0, \mathcal{O}, G)$, where $G = \{\text{on}(b, c)\}$.

Let $\mathbf{P} = (S_0, \mathcal{O})$ be a planning domain, α be an operator in $\mathcal{O}$ whose name is $\alpha(X_1, \ldots, X_n)$, and θ be a substitution that assigns ground terms to each $X_i, 1 \le i \le n$. Suppose that the following conditions hold:

$$\begin{aligned} &\{A\theta | A \text{ is an atom in } \text{Pre}(\alpha)\} \subseteq S; \\ &\{B\theta | \neg B \text{ is a negated literal in } \text{Pre}(\alpha)\} \cap S = \emptyset; \\ &S' = (S - (\text{Del}(\alpha)\theta)) \cup (\text{Add}(\alpha))\,\theta. \end{aligned}$$

Then we say that α is θ-*executable* in state S, *resulting* in state S'. This is denoted symbolically as $S \overset{\alpha,\theta}{\Longrightarrow} S'$.

Suppose $\mathbf{P} = (S_0, \mathcal{O})$ is a planning domain and G is a goal. A *plan that achieves* G is a sequence $S_0, \ldots, S_n, S_{n+1}$ of states, a sequence $\alpha_0, \ldots, \alpha_n$ of planning operators, and a sequence $\theta_0, \ldots, \theta_n$ of substitutions such that

$$S_0 \overset{\alpha_0,\theta_0}{\Longrightarrow} S_1 \overset{\alpha_1,\theta_1}{\Longrightarrow} S_2 \cdots \overset{\alpha_n,\theta_n}{\Longrightarrow} S_{n+1} \qquad (1)$$

and G is satisfied by S_n, i.e. there exists a ground instance of G that is true in S_n. The *length* of the above plan is n.

We will often say that (1) above is a plan that achieves G.

Let $\mathbf{P} = (S_0, \mathcal{O})$ be a planning domain or $\mathbf{P} = (S_0, \mathcal{O}, G)$ be a planning problem; and let $\mathcal{L}$ be the language of $\mathbf{P}$. Then:

1. $\mathcal{O}$ (and thus $\mathbf{P}$) is *positive* if for all $\alpha \in \mathcal{O}$, $\text{Pre}(\alpha)$ is a finite set of *atoms* (i.e. negations are not present in $\text{Pre}(\alpha)$).
2. $\mathcal{O}$ (and thus $\mathbf{P}$) is *deletion-free* if for all $\alpha \in \mathcal{O}$, $\text{Del}(\alpha) = \emptyset$.
3. $\mathcal{O}$ (and thus $\mathbf{P}$) is *context-free* if for all $\alpha \in \mathcal{O}$, $|\text{Pre}(\alpha)| \le 1$, i.e., $\text{Pre}(\alpha)$ contains at most one atom.

4. $\mathcal{O}$ (and thus $\mathbf{P}$) is *side-effect-free* if for all $\alpha \in \mathcal{O}$, $|\text{Add}(\alpha) \cup \text{Del}(\alpha)| \leq 1$, i.e., α has at most one postcondition.
5. $\mathcal{L}$ (and thus $\mathbf{P}$) is *function-free* if $\mathcal{L}$ contains no function symbols.

PLAN EXISTENCE is the following problem: "given as input, a planning problem $\mathbf{P} = (S_0, \mathcal{O}, G)$, is there a plan in $\mathbf{P}$ that achieves G?" It is important to note that in this formulation, planning is undecidable if there is no algorithm that halts on *all* possible input values of S_0, $\mathcal{O}$ and G. The fact that certain specific planning *domains* are decidable does not mean that the general planning problem, where arbitrary planning domains may occur in the input, is decidable.

3 Decidability and Undecidability Results

In this section, we show that logic programming is essentially the same as planning without delete lists. This is established by showing how to transform a deletion-free planning domain into a logic program such that for all goals G, the goal G is achievable from the planning domain iff the logical query that G represents is provable from the corresponding logic program.[2] As a consequence of this equivalence, we can use results on the complexity of logic programs and deductive databases to demonstrate the following results:

- PLAN EXISTENCE is *undecidable*, even if all operators have at most one precondition, and regardless of whether the operators have delete lists and/or negative preconditions.
- PLAN EXISTENCE is *decidable* if our first-order language contains no function symbols and is finitely generated (in particular, this means that only finitely many constants are present in the language). The presence or absence of delete lists does not affect the decidability result.
- When our planning domain $\mathbf{P} = (S_0, \mathcal{O})$ is fixed in advance, then the problem "given goal G, does there exist a plan that achieves G?" may still be undecidable depending on $\mathbf{P}$. The presence or absence of delete lists does not affect this result.

We now proceed to establish the equivalence between logic programming and planning without delete lists. Subsequently, we show how to do away with delete lists when function symbols are absent.

If $\mathbf{P}$ is deletion-free, then the *logic program translation* of an operator $\alpha \in \mathcal{O}$, denoted by $\text{LP}(\alpha)$, is the set of clauses

$$\text{LP}(\alpha) = \{(\forall)(A \leftarrow B_1 \,\&\, \ldots \,\&\, B_n) \mid A \in \text{Add}(\alpha)\},$$

where $\text{Pre}(\alpha) = \{B_1, \ldots, B_n\}$. The *logic program translation* of $\mathbf{P} = (S_0, \mathcal{O})$, denoted $\text{LP}(\mathbf{P})$, is the set of clauses

$$\text{LP}(\mathbf{P}) = S_0 \cup \bigcup_{\alpha \in \mathcal{O}} \text{LP}(\alpha).$$

Remark 1 Note that if we consider planning domains $\mathbf{P} = (S_0, \mathcal{O})$ where S_0 is infinite, then $\text{LP}(\mathbf{P})$ would contain infinitely many unit clauses. The infinite nature of $\text{LP}(\mathbf{P})$ will turn out to be irrelevant when $\mathbf{P} = (S_0, \mathcal{O})$ contains no operators that have negations in their preconditions. This irrelevance is due to the compactness theorem for first-order logic.

Lemma 1 Suppose that $\mathbf{P} = (S_0, \mathcal{O})$ is any *positive, deletion-free* planning domain, and

$$S_0 \stackrel{\alpha_0,\theta_0}{\Longrightarrow} S_1 \stackrel{\alpha_1,\theta_1}{\Longrightarrow} S_2 \cdots \stackrel{\alpha_n,\theta_n}{\Longrightarrow} S_{n+1}$$

is a plan that achieves some goal G (we really don't care what G is as far as this lemma is concerned). Then:

1. $S_0 \subseteq S_1 \subseteq S_2 \cdots \subseteq S_{n+1}$.
2. If operator α is θ-executable in state S_j, then α is θ-executable in state S_k for all $k \geq j$.

If $\mathbf{P} = (S_0, \mathcal{O})$ is a positive deletion-free planning domain, then $\text{LP}(\mathbf{P})$ is a *pure* (i.e., negation-free) logic program. The following theorem shows that achievability of a goal G in $\mathbf{P}$ is identical to provability of G from $\text{LP}(\mathbf{P})$.

Theorem 1 (Equivalence Theorem) Suppose $\mathbf{P} = (S_0, \mathcal{O})$ is a positive, deletion-free planning domain and G is a goal. Then there is a plan to achieve G from $\mathbf{P}$ iff $\text{LP}(\mathbf{P}) \models G$.

Corollary 1 (Semi-Decidability Results)

1. $\{G \mid G$ is an existential goal such that there is a plan to achieve G from $\mathbf{P} = (S_0, \mathcal{O})\}$ is a recursively enumerable subset of the set of all goals.

[2] This should be intuitively true, anyway, but the formal establishment of this equivalence is necessary before attempting to apply results from logic programming and deductive databases to planning problems. An important point to note is that we will only be considering truth in *Herbrand* models (cf. Shoenfield 1967) in this paper. It doesn't make much sense to consider non-Herbrand models for planning problems because the domains of such models often contain objects that do not occur in the language. In the case of blocks world, for instance, this corresponds to assuming (inside the model) that there are blocks on the table that cannot be referred to in the language. Obviously, this is not relevant to planning. Thus, when we talk of logical consequences of programs, we will be referring to those sentences that are true in all Herbrand models of the program. For function-free languages, this condition is well known to yield decidability of logical consequence (Plaisted 1984, Vardi 1982).

2. Given any recursively enumerable collection X of ground atoms (which, of course, are goals), there is a positive deletion-free planning domain $\mathbf{P} = (S_0, \mathcal{O})$ such that $\{A \mid A$ is a ground atom such that there is a plan to achieve A from $\mathbf{P}\} = X$

3. If we restrict $\mathbf{P}$ to be positive and deletion-free, then PLAN EXISTENCE is strictly semi-decidable.

Corollary 2 The problem "given a positive deletion-free planning domain $\mathbf{P} = (S_0, \mathcal{O})$, is the set of goals achievable from $\mathbf{P}$ decidable?" is Π^0_2-complete.

Corollary 3 If we restrict $\mathbf{P}$ to be positive, deletion-free, and context-free, then PLAN EXISTENCE is still strictly semi-decidable.

Corollary 4 Suppose $\mathbf{P} = (S_0, \mathcal{O})$ is a fixed positive, deletion-free planning domain. Then the problem: "given a goal G, does there exist a plan to achieve G?" is decidable iff the set of goals provable from LP($\mathbf{P}$) is decidable.

Theorem 1 holds only when $\mathbf{P}$ is positive. The reason for this is that if $\mathbf{P}$ is not positive, then LP($\mathbf{P}$) is a logic program that may contain negation in its body. Logic programming interprets negation in LP($\mathbf{P}$) as "failure to prove", which is different than the interpretation of negation in the planning domain $\mathbf{P}$. Intuitively, negation in logic programming says "conclude $\neg p$ if it is impossible to prove p". The corresponding notion of negation in planning would be "conclude $\neg p$ if there is no plan to achieve p" which is much stronger than saying "p is false in the current state." Thus, if $\mathbf{P}$ is not positive, then in some cases G will be achievable in $\mathbf{P}$ but LP($\mathbf{P}$) $\models G$ will be false. To see this, consider the following example:

Example 2 Consider the planning domain $\mathbf{P} = (S_0, \mathcal{O})$ that contains the two operators α_1, α_2 described below:

$$\begin{aligned} \text{Pre}(\alpha_1) &= \{\neg b\}, & \text{Add}(\alpha_1) &= \{a\} \\ \text{Pre}(\alpha_2) &= \{c\}, & \text{Add}(\alpha_2) &= \{b\} \end{aligned}$$

Suppose our initial state is the state $\{c\}$. Clearly, there is a plan to achieve a by simply executing operation α_1 in the initial state.

Now consider LP($\mathbf{P}$), which is the logic program:

$$\begin{aligned} a &\leftarrow \neg b \\ b &\leftarrow c \\ c &\leftarrow \end{aligned}$$

The set of atoms provable from this program according to logic programming (all major semantics for logic programs agree on this) is $\{b, c\}$, i.e. a cannot be obtained even though our planning domain admits a plan that achieves a.

Lemma 2 Suppose I is a decidable Herbrand interpretation, i.e. I is a decidable set of ground atoms, and G is a goal. Then

1. The problem "is goal G true in interpretation I?" is semi-decidable.

2. If the language $\mathcal{L}$ is known to be function free, then the above problem is decidable.

Theorem 2 (Decidability of Deletion-Free, Function-Free Planning) If we restrict $\mathbf{P}$ to be deletion-free and function-free, then PLAN EXISTENCE is decidable.

Theorem 3 (Theorem on Infinite Initial States) PLAN EXISTENCE is semi-decidable if we restrict $\mathbf{P} = (S_0, \mathcal{O})$ to be positive and deletion-free, and S_0 to be a *decidable* set of ground atoms of language $\mathcal{L}$ (even though S_0 may be infinite).

We now show that when $\mathcal{L}$ contains no function symbols, we can do away with delete lists. The idea is intuitively the same as that of Green (Green 1969, Nilsson 1980) (vis-a-vis the famous "Green's formulation of planning"), with one difference: Green introduces function symbols even if the original language contained none: we introduce new constants. When the language is function-free, only finitely many new constants are included.

Theorem 4 (Eliminating Delete Lists) Suppose $\mathbf{P}$ is a function-free planning domain. Then there is a positive deletion-free planning domain $\mathbf{P'} = (S'_0, \mathcal{O}')$ such that for each goal $G \equiv (\exists)(A_1 \& \ldots \& A_n)$, there is a goal $G' \equiv (\exists)(A'_1 \& \ldots \& A'_n \& \text{poss}(S))$, where "poss" is a new unary predicate symbol and for all $1 \leq i \leq n$, if $A_i \equiv p(t_1, \ldots, t_n)$, then $A'_i \equiv p(t_1, \ldots, t_n, S)$ where S is a variable symbol. Furthermore, G is achievable from $\mathbf{P}$ iff G' is achievable from $\mathbf{P'}$.

An important point to note is that even though delete lists may be removed, the size of $\mathbf{P'}$ is much larger than $\mathbf{P}$.

Theorem 5 (Decidability of Function-Free Planning) If we restrict $\mathbf{P}$ to be function-free, then PLAN EXISTENCE is decidable.

4 Chapman's Undecidability Results

To date, the best-known results on decidability and undecidability in planning systems are those of (Chapman 1987). However, there is a certain amount of confusion about what Chapman's undecidability results actually say, because some of his assumptions become clear only after a careful reading of the paper. To clarify the meaning of Chapman's undecidability results, we now compare and contrast his results with ours.

First Undecidability Theorem. Chapman's first undecidability theorem (Chapman 1987, pp. 370–371) says that all Turing machines with their inputs may be encoded as planning problems in the TWEAK representation, and hence planning is undecidable. Our results compare and contrast with this result in the following respects:

1. Chapman assumes that "an infinite set of constants t_i are used to represent the tape squares" (Chapman 1987, p. 371), but in his discussion of the result, he points out that this set of constants is recursive.

   Our language $\mathcal{L}$ contains only finitely many constants, but it may contain function symbols. This is essentially the same as having a recursive set of constants, because the function symbols can be used to generate countably many ground terms. Thus in this respect, our result doesn't differ from Chapman's first undecidability theorem.

2. Chapman uses an infinite initial state to prove his result. In particular, "there must be countably many successor propositions to encode the topology of the tape (and also countably many contents propositions to make all but finitely many squares blank)" (Chapman 1987, p. 371). He also says (Chapman 1987, p.344):

   > This result is weaker than it may appear ... the proof uses an infinite (though recursive) initial state to model the connectivity of the tape. It may be that if problems are restricted to have finite initial states, planning is decidable. (This is not obviously true though. There are infinitely many constants, and an action can in effect "gensym" one by referring to a variable in its post-conditions that is not mentioned in its preconditions.)

   We now describe how our results solve the open problem posed in the above quote.

   First, our Corollary 1 shows that if the language contains infinitely many ground terms, then planning is undecidable even if initial states are finite. This is true regardless of whether the infinite number of ground terms occurs because infinitely many constants are present in the language as is the case with (Chapman 1987), or because there are finitely many constants together with finitely many function symbols, as is true in our case.

   Second, our result Theorem 5 shows that if the language contains only finitely many ground terms, then planning is decidable. Planning with only finitely many ground terms in the language makes a lot of sense, because it applies to all domains where a finitely bounded number of "entities" are being manipulated (the usual formulation of the blocks world is one such example). We can make the number of these entities large, but as long as we keep it finite, these results apply.

3. Our Corollaries 1 and 3 demonstrate that planning with infinitely many terms is undecidable in general even if every planning operator contains no delete lists, at most one positive precondition, and no negative preconditions. Chapman's result (Chapman 1987, p. 371) is more restrictive: he needs four preconditions, and he explicitly uses negative post-conditions, which is equivalent to having delete lists.

Second Undecidability Theorem. The statement of Chapman's second undecidability theorem is that "planning is undecidable even with a finite initial state if the action representation is extended to represent actions whose effects are a function of their input situation" (Chapman 1987, p. 373). From his discussion on p. 371, it appears that Chapman is referring to the following kind of conditional operator:

Name: $\alpha(...)$
if P_α **then** add A_α and delete D_α
else add A'_α and delete D'_α

where P is a set of literals, and A_α, D_α, A'_α, and D'_α are sets of atoms.

A careful reading of Chapman's proof makes it clear that there is an additional assumption. In his proof, he makes use of operators that increment and decrement two counters, but there is no upper bound on the value of those counters—and thus it is necessary that the language contain infinitely many terms. But if there are infinitely many terms, then our Corollary 1 shows that planning is undecidable even with ordinary STRIPS-style planning operators.

On the other hand, suppose that the language contains only finitely many terms, but that there are conditional operators of the kind described above. Let α be any one of these conditional operators, and suppose that $P_\alpha = \{p_1, p_2, \ldots, p_n\}$. We now define an equivalent set of STRIPS-style operators $\{\alpha_N | N \subseteq P_\alpha\}$, none of which has conditional effects.

$\alpha_\emptyset$ is the following operator:

Name :	$\alpha_0(\ldots)$
Pre :	P_α
Add :	A_α
Del :	D_α

For every nonempty set $N \subseteq P$, α_N is the following operator:

Name :	$\alpha_i(\ldots)$
Pre :	$\{\neg p_i | p_i \in N\} \cup (P - N)$
Add :	A'_α
Del :	D'_α

The set of "unconditional" operators $\{\alpha_N | N \subseteq P\}$ is equivalent to α, in the sense that $S \stackrel{\alpha,\theta}{\Longrightarrow} S'$ if and only if there is exactly one α_i that is θ-executable in S, and $S \stackrel{\alpha_i,\theta}{\Longrightarrow} S'$.

Replacing each conditional operator by the equivalent set of unconditional operators produces a planning domain whose language has finitely many ground terms—and according to Theorem 5, planning is decidable for such domains.

From the above, it follows that the statement of Chapman's Second Undecidability Theorem is misleading. His proof of undecidability has nothing to do with whether the operators are conditional—it instead depends on the fact that his planning domain requires an infinite number of terms.

5 Conclusion

The primary aim of this paper has been to examine the decidability of planning with STRIPS-style planning operators (i.e., operators comprised of preconditions, add lists, and delete lists).

Our primary result is that planning is decidable if and only if the language has finitely many ground terms. This relates in several ways to Chapman's work (Chapman 1987):

1. It solves an open problem posed in (Chapman 1987), regarding the decidability of planning with a finite initial state. The answer depends on whether the language has finitely many ground terms.
2. It shows that one of the results in (Chapman 1987) is misleading. The undecidability of planning with STRIPS-style operators has nothing to do with the presence or absence of conditional effects. If the language has finitely many ground terms, then planning is decidable even if the operators have conditional effects. If the language has infinitely many ground terms, then planning is undecidable even if no operators have no conditional effects, no delete lists, and no negative preconditions.

References

H.A. Blair. "Canonical Conservative Extensions of Logic Program Completions," *Proc. 4th IEEE Symposium on Logic Programming*, 1989, pp. 154–161.

Eugene Charniak and Drew McDermott. *Introduction to Artificial Intelligence*. Addison-Wesley, Reading, MA, 1985.

D. Chapman. "Planning for Conjunctive Goals," *Artificial Intelligence* **32**, 1987, pp. 333-377.

K. Erol, D. S. Nau, and V. S. Subrahmanian. "Complexity, Decidability and Undecidability Results for First-Order Planning," submitted for journal publication, 1991.

C. Green. "Application of Theorem-Proving to Problem Solving," *Proc. IJCAI-69*, 1969.

Naresh Gupta and Dana S. Nau. "Complexity results for blocks-world planning," *Proc. AAAI-91*, 1991. Honorable mention for the best paper award.

Naresh Gupta and Dana S. Nau, *Artificial Intelligence*, accepted for publication, 1992.

N. J. Nilsson 1980. *Principles of Artificial Intelligence*. Tioga, Palo Alto, 1980.

Earl D. Sacerdoti. "Planning in a hierarcy of abstraction spaces." In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 98–108. Morgan Kaufman, 1990. Originally appeared in *Artificial Intelligence* **5** (1974), 115–135.

D.A. Plaisted. "Complete Problems in the First-Order Predicate Calculus," *Jour. Computer and Systems Sciences* **29** (1984), pp. 8–35.

Earl D. Sacerdoti. "The nonlinear nature of plans." In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 206–214. Morgan Kaufman, 1990. Originally appeared in *Proc. IJCAI-75*.

J. Sebelik and P. Stepanek. "Horn Clause Programs for Recursive Functions." In K. Clark and S.-A. Tarnlund, editors, *Logic Programming*, pp. 325–340, Academic Press, 1980.

J. Shoenfield. *Mathematical Logic*, Academic Press, 1967.

A. Tate, J. Hendler, and M. Drummond. "A review of AI planning techniques." In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 26–49. Morgan Kaufman, 1990.

M. Vardi. "The Complexity of Relational Query Languages," *Proc. 14th ACM Symp. on Theory of Computing*, San Francisco, 1982, pp. 137–146.