This paper was selected by a process of
anonymous peer reviewing for presentation at

COMMONSENSE 2007

8th International Symposium on Logical Formalizations of Commonsense Reasoning

Part of the AAAI Spring Symposium Series, March 26-28 2007,
Stanford University, California

Further information, including follow-up notes for some of the
selected papers, can be found at:

www.ucl.ac.uk/commonsense07

Learning a Plan in the Limit

Patrick Caldon and **Eric Martin**
School of Computer Science and Engineering
The University of New South Wales
UNSW 2052, NSW, Australia

Abstract

Induction is one of the major forms of reasoning; it is therefore essential to include it in a commonsense reasoning framework. This paper examines a class of induction-based planning problems. These problems can be solved, but not in the classical sense, where one and only one output—a correct solution—is to be provided. Here a sequence of outputs, or hypothetical solutions, is output, with finitely of them being possibly incorrect, up to a point where the sequence becomes constant, having converged to a correct solution. Still it is usually not possible to ever discover *when* convergence has taken place, hence it is usually not possible to definitely *know* that the problem has been solved, a behavior known in the literature on inductive inference, or formal learning theory, as *identification in the limit*. We present a semantics for iterative (looping) planning based on identification in the limit, with the planner learning from positive examples (as opposed for instance to negative examples, or answers to queries). Potential plans are repeatedly hypothesized, and a correct plan will be converged upon iff such a plan exists. We show that this convergence happens precisely when a particular logical entailment relation is satisfied. The existing system KPLANNER, which generates iterative plans by generalizing over two examples, is analogous to a fragment of this procedure. We describe an optimizing version, which attempts to satisfy a goal in the best possible way by defining preferences on plans, in the form of an order relation. We also discuss potential extensions to planning which cannot be solved by induction (and a fortiori, cannot be solved in the classical sense).

Introduction

Objective of this work

The objective of this work is to explore the technical underpinnings of a particular kind of commonsense reasoning activity where an agent is trying to determine appropriate behaviour based on empirical knowledge about the world. Think of how someone develops the skills for a complex technical activity. For an example, consider boating, but many activities have the same character.

A boatman necessarily has operational knowledge about the various activities involved in boating. He knows that if the boat has a particularly heavy load, water will start lapping over the edge, and also that the engine will consume more fuel. He knows that travelling quickly makes the boat harder to turn. There are many specialized pieces of knowledge of this kind which are required in order to successfully operate a boat.

Now consider how these pieces of knowledge can be acquired. One might determine the maximum load for a boat by examining the geometry of the hull, the weight of the boat, the density of water, and applying physical theories. But it is also possible to gain this knowledge empirically, for instance by observing circumstances under which a similar boat is overloaded, or circumstances under which a fast moving boat turns more slowly. This could be done by trying something, making a mistake, and noting that that ought not be done again; or it could be done by hearing the story of someone who made a mistake. For instance a boatman might one day slightly overload the boat, and notice that it sits too low in the water, that too much water ships onto the boat, and then change the actions taken in loading the boat to ensure that this does not eventuate again. In this way observations of failed attempts at action are used to guide future action. In practice humans often acquire this kind of knowledge empirically, rather than by reasoning from the first principles of physics. Clearly, empirical reasoning is an important component of commonsense reasoning.

The kind of commonsense reasoning activity that we have illustrated involves elements of many existing frameworks. Reasoning about action, with an underlying non-monotonic inference relation, is a key component of the scenario. For instance, the boatman may perform actions where the accumulated experience thus far leads him to believe the actions are safe, but discover on performing them that they are actually unsafe; in this way new experience can cause the inference made on the basis of existing experience to be contradicted. The learning involved in this scenario is a kind of non-monotonic reasoning. Our aim is to put all these pieces together, propose a framework for learning how to generate correct plans.

Limiting planning

Reiter draws a distinction between closed and open world versions of planning (Reiter 2001), to be both contrasted with our proposed *limiting planning*.

- In closed world planning, facts concerning the initial situation are provided before the planner starts, and any po-

tential fact that is not provided is assumed not to hold in the initial situation.

- In open world planning, facts concerning the initial situation are provided before the planner starts, but there is no assumption concerning any of the potential facts that are not mentioned.
- In limiting planning, an *enumeration* of fluent values for all possible initial situations is provided while the planner runs; all true facts about the initial situation are guaranteed to be eventually provided to the planner.

The non-terminating scenario inherent to limiting planning might seem to be an unnecessary complication, but we will show that it is an *essential* feature. Indeed, we will see that while it is *impossible* to discover plans for the natural class of problems that we describe if the set of facts to be provided is computed in advance, it is still possible to discover plans if the set of facts to be provided is allowed to be arbitrarily large. Only finitely many facts will be needed to output a plan, but which facts are needed, or how many facts are needed, cannot be known in advance. So we consider planning problems for which a solution can be found in finite time, but for which it cannot be known that a solution has been found. Hence limiting planning is not a variation on closed world planning; we cannot simply wait until sufficiently many facts are provided to the planner, and apply closed world planning, as we have no way, in general, to convert "sufficiently many" into a numeric quantity.

How do we know if planning is successful? In conventional planning, where the initial situation, or collection of initial situations, is completely specified at the start of the planning activity, success is equivalent to producing an object which, if suitably interpreted, causes the agent to always (or in some variants, sometimes) reach the goal. But we focus on planning tasks that go beyond what classical planning can achieve, we look at planning activity that inherently involves induction, so we need something more than just "the goal must be reached by the plan." Here we take our cue from formal learning theory, and say that planning is successful if the potential-plans (hypothesized plans which might or might not work) **stabilize** to some particular term which reaches the goal—we call this term a *correct plan*. If the agent were never again to change its mind then it would be successful under a notion which is exactly analogous to the notion of *identification in the limit* from formal learning theory.

We see the key contribution of this paper as the description of a planning by induction framework, formalized with the help of a particular notion of logical entailment. Further contributions include the development of the previously unpublished optimizing limiting resolution procedure, and the descriptions of limitations of similar kinds of framework. We do have a proof-of-concept implementation of a limiting planner, but the focus for this paper is the semantics; implementation is outside the scope due to space constraints.

A Motivating Example

A concrete example is useful to give the idea of the kinds of problems being addressed and the method for addressing them.

Example 1. *An agent is periodically given a bag of blocks of unknown size, and must rearrange all the blocks into at least two piles of size at least two.*

This is a straightforward statement of an initial situation for a plan and the goal of a plan. The planner is permitted to use iterative expressions of the goal, that is containing loops. This can be encoded using Levesque's specification for robot plans (Levesque 1996) and a few actions; the examples below give the flavor. Here `empty` is a sensing action to see if there is a block left in the bag, and `put(k)` puts a block into pile `k`, so for instance `put(2)` would place a block on pile number 2. The at-least-two restriction removes some trivial solutions.

Imagine now an agent (who believes the world is regular) trying to solve this problem by induction (i.e. generalizing) over examples. The agent is shown 20 blocks all on the table, numbered $1, ..., 20$. Suppose it somehow generates:

```
LOOP:
  put(1); put(2); put(3); put(4)
  BRANCH(empty):
     true: exit
     false: next
END LOOP
```

The interpretation of this plan corresponds to its natural reading; we put a block on each pile 1 through 4 until all the blocks in the bag are exhausted, and end up with 4 piles of size 5. Now the agent is given a bag with 40 blocks. Let's say it proposes the same plan again, which we can see also satisfies the condition.

Now suppose it is given a bag with 15 blocks; the above plan will not work, as it will produce 3 stacks of size 4 and one stack of size 3 and get itself into a situation where the prescribed action `put(4)` is not possible. The planner must re-plan and might produce:

```
LOOP:
  put(1); put(2); put(3); put(4); put(5)
  BRANCH(empty):
     true: exit
     false: next
END LOOP
```

Note that this term works for 15, 20 and 40 blocks, and so is general enough to cover all the examples thus far seen. This process continues indefinitely, with the planner being shown initial conditions and producing a plan.

Unlike conventional frameworks, there is no single initial situation, and the agent has to perform induction from a collection of example initial situations; it is having to learn. It repeatedly produces plans which satisfy a goal criteria from the initial situations thus far seen. It can use its theory of action to determine if a potential-plan works in a particular initial situation.

It is important to note that *iterative* plans (involving loops) are necessary if we are to generalize to piles of any size. Clearly, classical *sequential* plans (without loops or branches) are not sufficient for representing solutions to this problem. Furthermore, while *conditional* plans (with branches) can be produced for any finite collection of piles, they will not generalize to an infinite enumeration of piles.

An AGM-Based Account?

Now one potential option would be to employ some kind of belief revision framework. Let us try and apply an AGM based system to the above(Alchourron, Gardenfors, & Makinson 1985). An AGM system contains a belief revision operator $\star$ which takes a theory T and a formula φ as arguments, with $T \star \varphi$ denoting the theory obtained after T has been updated with respect to φ. The AGM conditions constrain the behavior of the $\star$ operator with eight conditions, for instance, that $T \star \varphi$ includes φ. Imagine we have somehow encoded our theory of piles in $\mathcal{D}_{PILE}$, and proceed using some kind of iterated belief revision; we might have to represent 15 blocks in the bag by $\varphi_1 = bagCount(15, s_1)$, indicating that in the first situation, there are 15 blocks in the bag. Then we might have to write $\varphi_2 = bagCount(20, s_2)$ to indicate that in the second situation there are 20 blocks in the bag. Our (revised) theory would then be $(\mathcal{D}_{PILE} \star \varphi_1) \star \varphi_2$. Now if the set $\mathcal{D}_{PILE} \cup \{\varphi_1, ...\}$ is consistent and we set our belief revision operator to the deductive closure of all the formulas thus far seen, we will never have to contract our knowledge base because the addition of new formulas never produces any inconsistency.

The problem comes when we consider what kinds of queries we wish to answer with this system. One natural possibility is queries of the form (abusing notation):

$$Axioms \cup \{\varphi_1, \varphi_2, ...\} \models^* \exists p\, [plan(p) \wedge \forall s_i\ initial(s_i) \rightarrow works(p, s_i)]$$

This should read: is there some plan such that for all possible initial situations, the plan works? We might hope that there is some collection of postulates K, consistent with $Axioms \cup \{\varphi_1, \varphi_2, ...\}$, that we can add to $Axioms$ so that $(K \cup Axioms \star \varphi_1) \star \varphi_2 ... \models \varphi$ is equivalent to $Axioms \cup \{\varphi_1, \varphi_2, ...\} \models^* \varphi$, with φ being the entailed formula in the above relation. We can make finitely many mistakes in stacking our piles, but there should be an l such that, taking an arbitrary $k \geq l$ and substituting $\{\varphi_1, ..., \varphi_k\}$ in the above relation for $\{\varphi_1, \varphi_2, ...\}$, the resulting relation holds (and actually implies the existence of a witness for the quantified variable p, that codes a working plan). Still, under an AGM update policy, our agent cannot draw any conclusion which requires him to generalize over all possible initial situations (the $\forall s_i ...$ part of the query), so he will never produce a working plan. [1] This feature is natural since the AGM postulates do not make any reference to the quantificational structure of formulas.

Towards a Formalization

Identification

We remind the reader of the fundamental concepts of inductive inference (Gold 1967; Jain *et al.* 1999). Suppose we have some collection of countable sets $\mathcal{L}$. Given $L \in \mathcal{L}$, call a sequence e of elements of $L \cup \{\sharp\}$ [2] such that every member of L is in e, an *enumeration* of L. Use $SEQ_{\mathcal{L}}$ for the collection of finite initial segments of enumerations of members of $\mathcal{L}$. A *learner* is a partial function $f : SEQ_{\mathcal{L}} \rightarrow \mathbb{N}$, where $\mathbb{N}$ is seen as an index set for $\mathcal{L}$ (a set of codes for the members of $\mathcal{L}$). A learner is said to *identify* some $L \in \mathcal{L}$ if for every enumeration e of L, it reports the index of L in response to all but a finite number of finite initial segments of e. The collection $\mathcal{L}$ is said to be *identifiable* if there is some learner which identifies every member of $\mathcal{L}$. Note that if $\mathcal{L}$ is a singleton, identification becomes trivial; when $\mathcal{L}$ is larger the problem becomes more substantial.

Visualize this scenario as playing a game where you must guess a set that we have in mind. Suppose we decide to play with $\mathcal{L}$ as the collection of sets consisting of all the numbers greater than some natural number (an example for L is $\{5, 6, 7, \dots\}$). Having first chosen (for example) $\{2, 3, 4, \dots\}$, we then proceed through the game in rounds of guesses, as follows. Suppose we tell you 4 is in the set; you might guess $\{4, 5, 6, \dots\}$ as the set. If we tell you 7 is in the set, then presumably you would decide to stick with your guess. If however we tell you next that 2 is in the set, you might want to change your mind and revise your guess to $\{2, 3, 4, \dots\}$. As from this point on we will never mention the number 1, you'll always be happy with this guess of $\{2, 3, 4, \dots\}$. But since you've only ever seen a finite initial sequence, it will appear to you that you *may* have to revise your guess at any time, and for this set L there's no finite collection of data which allows you to be sure that your guess is correct, and so you will not have a property analogous to compactness in classical logic. It is clear, however, that the strategy of "guess the set whose least element is the lowest number thus far presented" will, *in the limit*, produce the correct hypothesis, that is, after at most a finite number of mind changes on your part.

Situation Calculus

The situation calculus is a formalism which provides a systematic methodology for expressing actions, and encompasses problems in conditional planning; we present a simplified summary here. [3] The only concrete difference between this and conventional expositions is that we require countably many initial situations producing a forest of isomorphic situation trees rooted at these initial situations, but we do not introduce a modal operator or K relation.The situation calculus assumes a first-order vocabulary, with symbols classified as follows. Terms of the form $do(\alpha, s)$ are distinguished as *situations*, where s is itself a situation, and $s_0(i)$ is an *initial situation*, one for each $i \in \mathbb{N}$ (using $\mathbb{N}$ is just for convenience); this generalizes the singleton s_0 used in the standard developments of the framework. Here α is an *action* term, and $do(\alpha, s)$ is the situation obtained by performing action α in situation s. All constants which are not actions or situations are *objects*. *Fluents* are predicates of the form e.g., $ontable(b_1, s)$, with a situation only in the final argument; this would represent that b_1 is on the table in

[1] This is a consequence of the AGM postulate $T \star \varphi \subseteq \{\psi : T \cup \{\varphi\} \models \psi\}$

[2] The $\sharp$ symbol represents no data being provided to the learner.

[3] Many situation calculus details are glossed over here for space reasons, and suggestive examples are given rather than formal definitions; see Reiter's book (Reiter 2001), particularly chapter 4, for a complete exposition.

situation s—functional fluents are not considered in this account. The predicate $Poss(\alpha, s)$ is true iff a is possible in situation s.

A situation calculus theory is divided into collections of axioms. The *foundational axioms* Σ entail that the situations form a tree and that there are no non-standard situations; the *unique names axioms* guarantee that all actions are distinct, and identical actions have identical arguments; logic programming formalisms, particularly the restriction to Herbrand models, can be used to guarantee these are maintained (Baral & Gelfond 1999). A collection of *action precondition axioms* $\mathcal{D}_{ap}$ defines the predicate $Poss$, which enjoys a uniformity property ensuring that the preconditions for the executability of some action are determined solely by the current situation; these are of the form e.g.

$$\forall x, s(Poss(pickup(x), s) \leftrightarrow ontop(x, s) \wedge freehand(s))$$

Successor state axioms $\mathcal{D}_{ss}$ define what happens to the truth value of some fluent as some action is performed and are of the form e.g. $\forall a, s(handempty(do(a, s)) \leftrightarrow handempty(s) \wedge \forall x(a \neq pickup(x)))$. These similarly enjoy a uniformity property.

Given a language for situation calculus, we can define a *p-plan* as a term in Levesque's robot planning language (Levesque 1996). As noted, interactive plans are required if we are to perform induction over a collection of initial situations. This language is the closure of the following:

- $seq(a, p)$, for action a and p-plan p
- $branch(a, p_1, p_2)$ for action a and p-plans p_1 and p_2
- $loop(p_1, p_2)$ for p-plans p_1 and p_2
- $next$ and $exit$

Call this language $\mathbb{P}$. Below, we will define a *correct plan* as a p-plan which causes the goal to be reached.

Given a planning domain, we will not in general want to consider all possible initial situations; some of these initial situations will be degenerative, and we would expect a plan to fail for these. Some initial situations will be valid states for commencing the execution of a correct plan which we expect will lead to the goal, named *success initial situations*. For a given problem, these will be terms of the form $s_0(i)$, denoted $\mathbb{S}$. An *initial fluent value* is a potential value of a fluent in the initial situation, and will have the form $f(...., s_0(i))$.

Some planning axioms are also required; they are given in $\mathcal{P}$, which is a fragment of a logic program based on the robot program axiomatization (Levesque 1996). These axioms give precise conditions for a p-plan moving the state from one situation to another. This permits the clausal definition of a predicate $moveto(p, s_{\mathrm{i}}, s_{\mathrm{f}})$, with intended interpretation that the execution of p-plan p in situation s_{i} may lead to some situation s_{f}. We then check that the goal fluents have appropriate values in s_f to make the statement that the p-plan works. The following axioms for $\mathcal{P}$ are required; they are taken directly from (Levesque 1996):

$$\begin{array}{l}
P(seq(a,p), s_i, s_f, x) \leftarrow \\
\qquad Poss(a, s) \wedge P(p, do(a, s_i), s_f, x) \\
P(next, s, s, 1) \\
P(exit, s, s, 0) \\
P(branch(a, p_1, p_2), s_i, s_f, x) \leftarrow \\
\quad Poss(a, s) \wedge SF(a, s_i) \wedge P(p_1, do(a, s), s_f, x) \\
P(branch(a, p_1, p_2), s_i, s_f, x) \leftarrow \\
Poss(a, s) \wedge \neg SF(a, s_i) \wedge P(p_2, do(a, s), s_f, x) \\
P(loop(p_1, p_2), s_i, s_f, x) \leftarrow \\
\qquad P(p_1, s_i, s', 0) \wedge P(p_2, s', s_f, x) \\
P(loop(p_1, p_2), s_i, s_f, x) \leftarrow \\
\qquad P(p_1, s_i, s', 1) \wedge P(loop(p_1, p_2), s', s_f, x) \\
moveto(p, s_i, s_f) \leftarrow P(p, s_i, s_f, 1)
\end{array}$$

Informally, the predicate P makes the plan progress forward by one action.

Mapping Piles to Identification

Suppose we have the following axiomatization in the Piles domain as described informally in Example 1; for actions we have:

- sensing action $empty$ where $Poss(empty, s)$ is always true and $SF(empty, s) \leftrightarrow bagCount(0, s)$—we are "empty" if the bag of blocks is empty.
- $put(k)$ with $Poss(put(k), s) \leftrightarrow bagCount(n, s) \wedge n > 0$.

Here SF is an axiom for sensing actions, which gives the result of performing a sensing action at any time.

For successor state axioms we have:

- $bagCount(n-1, do(put(k), s)) \leftrightarrow bagCount(n, s)$
- $pileCount(k, n+1, do(put(k), s)) \leftrightarrow pileCount(k, n, s)$
- Frame axioms for the fluents

We now want to develop interactive plans which satisfy these conditions, called $\mathcal{D}_{PILE}$, plus some collection of initial situations.

Rather than having the initial situations being presented logically, we will present them using the following coding c, where $c(k)$ is defined as

$$\{bagCount(k, s_0(k))\} \cup \{pileCount(i, 0, s_0(k)) : i \in \mathbb{N}\}$$

Further suppose we have a function p where:

$$\begin{array}{l}
p(x) = loop(seq(put(1), ...seq(put(x), \\
\qquad branch(empty, exit, next))...)
\end{array}$$

with x many puts in the sequence. The examples in the discussion above are for $p(4)$ and $p(5)$.

Suppose we have a second-order axiom ψ to minimize the P predicate above. A plan will exist for a single instance of k if there is a number i satisfying $\Sigma \cup \mathcal{D}_{una} \cup \mathcal{D}_{PILE} \cup c(k) \cup \mathcal{P} \cup \{\psi\} \models \forall s_f\ moveto(p(i), s_0(k), s_f)$.

Suppose we set $L_k = \{k \cdot i : i \in \mathbb{N}\}$ and $\mathfrak{L} = \{L_k : k \in \mathbb{N}^+\}$. Then the following can easily be shown:

Proposition 2. *There is a learner f which identifies $\mathfrak{L}$.*

Having set $f(\langle i_1, ..., i_k \rangle) = \gcd(\{i_1, ..., i_k\})$, it is easy to verify that f is a learner for $\mathfrak{L}$. More importantly, the functions p and c (or its inverse) allow us to map a sequence of sets of initial fluent values to a plan for $\mathfrak{L}$. With $L'_i = \{c(i) : i \in \mathbb{N}\}$, it is clear that we can analogously identify L'_i from an enumeration, to produce an integer index for i, and then map this index using p to a plan for solving the planning problem above. The semantics of planning have become equivalent to the act of identifying a collection of initial situations. [4]

One possible complaint is that the limiting planning agent will never know that the plan is correct—correctness here is a limiting property. The usual expectation is that a planner outputs a single (correct) plan in response to any problem. For the natural class of problems we are considering, it is *impossible* to construct a (conventional) planner which produces only correct plans. The following propositions are straightforward consequences of learning theory and are derived from similar results in (Jain *et al.* 1999). They show that an identification-in-the-limit approach is necessary for solving problems in some planning domains, but incapable of solving all planning problems.

Suppose 0 is the index for the empty set. Define e_i as the ith element of enumeration e. In the manner of (Jain *et al.* 1999), define a *self-monitoring learner* as a learner f such that for all enumerations e, there exists a unique $n \in \mathbb{N}$ such that $f(e_n) = 0$ and for all $i > n$, $f(e_i) = f(e_{i+1})$. The idea is that 0 is used as a marker in the output indicating that the following values will be definitive.

Proposition 3. *No self-monitoring learner can identify* $\mathfrak{L}$.

The relevance of this is that it shows the limiting planning framework is distinct to the open-world and closed-word frameworks. One natural suggestion is to build a system which accumulates the initial part of the sequence as it becomes available, and use some mechanism m which takes this initial part of the sequence as its argument to determine when enough data is available. Once m reports that sufficient data is available to produce a plan, it hands that data over to a conventional planner. Using m and a conventional planner, we can however construct a self-monitoring learner for $\mathfrak{L}$, so no such m can exist. While this argument is presented informally here, formalization is not difficult.

It should also be clear from this analysis that the problem of finding a correct plan requires some kind of iterative planning for all but the most trivial cases. In particular, it is not possible to find a conditional plan which identifies $\mathfrak{L}$ above, but there is an iterative plan for every member of $\mathfrak{L}$.

Logical Framework

Representing Data

While all of this work is perhaps interesting, logic is not a first-class citizen in the above analysis; the situation calculus is just a formalism that encodes the problem. We now present a logical account. In this inductive framework, some formalism is required to represent sequences of initial situations, each of which intuitively corresponds to the direct observations an agent can make. Suppose we have some theory of action $\mathcal{D}$, then this sequence ought to have several features:

- it must be an *enumeration*, that is every initial situation must appear at some stage in the sequence,
- it must be consistent with $\mathcal{D}$,
- it must consist of literals

The restriction to literals makes sense given that the sequence is meant to represent primitive observations an agent can make. For instance, it seems intuitively reasonable that an agent can observe a directly that the initial situation has 15 blocks in the bag, but it does not seem reasonable that an agent can observe "all bags contain a multiple of 5 blocks". However, this statement does seem appropriate as background information, since it might be possible that our agent somehow knows this information a-priori.

In this way, we identify two sets of formulas, one being *possible data* $\mathbb{D}$, corresponding to possible observations, that is formulas which may appear in an enumeration, the other being *possible assumptions* $\mathcal{A}$, a collection of formulas from which we will draw background information. The observations will be a subset of $\mathbb{D}$. Suppose we have some vocabulary $\mathcal{V}$ for the situation calculus. To build a theory of action, the possible assumptions will be the collection of possible successor state and action precondition axioms over $\mathcal{V}$, with those axioms in $\mathcal{D}_{S_0}$ which do not describe initial fluent values.

The collection of possible data will be the collection of possible initial fluent values. For instance, in the piles example, it will be the collection of all positive instances of the $bagCount$ predicate, i.e. $\mathbb{D} = \{bagCount(n, s_0(i)) : n, i \in \mathbb{N}\}$.

We can form a subset of $\mathbb{D}$ from a structure $\mathfrak{M}$ by taking the collection of all the formulas of $\mathbb{D}$ true in $\mathfrak{M}$. Call this subset the $\mathbb{D}$-diagram of $\mathfrak{M}$, and write:

$$\mathrm{Diag}_{\mathbb{D}}(\mathfrak{M}) = \{\varphi \in \mathbb{D} : \mathfrak{M} \models \varphi\}$$

To constrain this further, we do not permit arbitrary possible worlds, but restrict to Herbrand structures. This makes the foundation axioms and the unique-names axioms of the situation calculus implicit, as well as ensuring that there are no non-standard elements corresponding to non-realizable potential plans. Using answer-set semantics, as do Baral and Gelfond (Baral & Gelfond 1999), would have a similar effect, however we would not have the limiting property in our semantics. This restriction causes the foundational axioms and the unique names axioms to be satisfied. There will also be some *domain theory* $\mathcal{D}$ which is a subset of $\mathcal{A}$, that is a collection of action preconditon and successor state axioms, alongside any axioms from $\mathcal{D}s_0$ which do not describe initial fluent values. For the class of intended models of a background theory, it is sensible to follow the example of various situation calculus frameworks and choose the class of Herbrand models of $\mathcal{D}$ as intended models.

With these elements in place, it is possible to describe a *planning problem*, which are the objects from which in-

[4] Note that this is a discussion of the semantics of planning; it is of course not being suggested that the g.c.d. operator is a general planning mechanism.

ference will be performed. A planning problem is a set of formulas of the form $\mathcal{D} \cup \mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$, where $\mathfrak{M} \models \mathcal{D}$ and each fluent has a unique value in each initial situation. This is then the domain theory with the collection of initial conditions an agent might encounter attached to it.

Define a $\mathbb{D}$-minimal model of $T = \mathcal{D} \cup \mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$ as a Herbrand model of T with a minimal $\mathbb{D}$-diagram under set inclusion. The relation $T \models^{\mathbb{D}} \varphi$ is read as "every $\mathbb{D}$-minimal model of T is a model of φ." Suffice it to say here that this logical consequence relation is stronger than classical logical consequence, so if $T \models \varphi$ then $T \models^{\mathbb{D}} \varphi$. Of course, the converse does *not* hold in general. Since this notion is not compact, it requires a more general notion of proof; identification forms the crux of this notion. The motivation and a discussion of the model theory of these semantics are in (Martin, Sharma, & Stephan 2002).

Using this, we can at last formally define an *l-planner* as a partial function from initial segments of enumerations of some $\mathbb{D}$ to the collection of p-plans.

Limiting Resolution

Limiting Resolution (LR) is a resolution system which provides a proof procedure sound and complete with respect to the above semantics ($\models^{\mathbb{D}}$). A complete exposition appears in (Caldon & Martin 2004), but an abbreviated description of the LR procedure is given here. A program (called a *limiting definite program*) is a form of general logic program, which includes negative $\mathbb{D}$-literals, the full details of which are explained in (Caldon & Martin 2004). A particular feature of a limiting definite program is the inclusion of an enumeration of some $\mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$ above. A *limiting definite query* Q is a closed formula of the form: $\exists \overline{x}(\psi(\overline{x}) \wedge \forall \overline{y} \neg \chi(\overline{x}, \overline{y}))$ (with additional restrictions on ψ and χ), and the LR procedure will construct a sequence of computed answer substitutions (c.a.s.) based on such a query and a limiting definite program. The query attempts to find a witness for $\overline{x}$ generated by ψ; χ can be seen as a test phase for the generated witness. Note that there is no requirement for ψ to be decidable.

Given an enumeration $\langle \varphi_1, \varphi_2, \dots \rangle$ of the $\mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$ component of some planning problem, the *LR procedure* is as follows:

```
Set i = 0; Set k = 0
Loop:
   Set P = B ∪ {φ_1, ..., φ_i}
   Attempt ψ over P producing {θ_1, ..., θ_i}
   For each witness θ_j attempt χ·θ_j over P:
      If χ·θ_j does not succeed
             within i steps for some j ≤ i,
        choose least j and place θ_j in S_i
   Increment i; If S_i ≠ S_{i-1} increment k
   Report (S_i, k)
End Loop
```

Here $\chi \cdot \theta$ is the formula produced from χ by applying the substitution θ. Wherever the algorithm states "attempt query," construct a SLD-resolution tree from the query ψ or $\chi \cdot \theta$ and theory P to produce a collection of unifiers in the standard fashion, except that negative $\mathbb{D}$-literals in the partial tree unify with negative $\mathbb{D}$-literals in the program. The output of the algorithm is the reported sequence of sets of computed answer substitutions, along with the number of mind changes so far: (S_i, k). Note that the procedure is nonterminating, so the sequence is infinite. If there is an upper bound on the number of mind changes k in the ordered pair, then the procedure is said to *succeed* for Q; otherwise it *fails*. If the procedure succeeds, then the successful witness is the substitution corresponding to this upper bound. The procedure is sound and complete, that is the procedure will succeed for some limiting definite program P and query Q iff $P \models^{\mathbb{D}} Q$. There are clear similarities with SLDNF, however unlike SLDNF, we replace finite failure with a limiting (i.e. non-finite) version of failure.

Planning Problems

Suppose that $\mathcal{D}$ can be expressed in a limiting definite program. Note that by using a subset of the Lloyd-Topor transformations, the axiomatization for Example 1 can be readily transformed into a limiting definite program.

On account of the use of a minimal model semantics, we could re-express this theory as its Clark completion. Negation occurs in the above theory, in $\neg SF$ and in inequality which will appear in the frame axioms; the existence of this precludes the theory in its above form from being a limiting definite program. However, in the context of solving planning problems, the value SF should be completely determined by the (finite) collection of initial fluent values corresponding to an initial situation and the successor state axioms. Further inequality can be expressed positively under the assumption of a finite vocabulary. If this condition holds in any enumeration of $\mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$, all positive fluent values for the initial situation will eventually appear, enabling us to use negation-as-failure to determine the value and SF. Note that here none of the axioms require an explicit modal operator, which is defined in terms of a successor state axiom for K in (Levesque 1996). Further unlike this work, since we are restricted to Herbrand structures in the logical consequence relationship and limiting definite programs, we will necessarily have the minimal such P and do not need an explicit second-order axiom to guarantee the minimal such P. As per Levesque's comments, the distinction between $next$ and $exit$ is that inside a $loop(r_i, r_t)$, the former causes r_i and then the loop to be executed, whereas the latter causes r_t to be executed.

Thus far this is close to the conventional development of planning using the situation calculus. We do not however require a closed initial database; instead $\mathbb{D}$ is defined as the collection of predicates of the form $f(v, s_0(i))$, with the intended interpretation that fluent f has value v in situation $s_0(i)$. Choosing a particular Herbrand model $\mathfrak{M}$ of $\mathcal{D}$ is equivalent to setting the initial fluent values for the problem at hand.

We choose the collection of initial situations for which we intend our (correct) plan to succeed by enumerating all the members of $\mathbb{D}$ true in some structure $\mathfrak{M}$. The set $\mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$ will take the role of $\mathcal{D}_{s_0}$ in more conventional developments of the planning using the situation calculus. Note we don't need any additional axioms in $\mathcal{D}_{s_0}$, but it might be convenient to add them in practice for performance reasons. For instance, we might want to add $pileCount(x, 0, s_0(i))$, but

this is not required.

Thus in a particular domain, every planning problem is identified with a particular subset of $\mathbb{D}$ which is consistent with the background knowledge. The background knowledge will specify a collection of Herbrand models. A particular, such structure $\mathfrak{M}$ will be selected (as a particular instance of the problem, this instance being a priori unknown to the planner), and the 1-planner will receive an enumeration of $\mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$. If a solution exists, the 1-planner will be able to identify the particular $\mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$ of interest. A byproduct of that identification is a plan. Putting all this together, the program will be of the form $\mathcal{D} \cup \mathcal{P} \cup \mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$. Note $\mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$ can be infinite.

For example, suppose $\mathfrak{M}$ is an interpretation of $\mathcal{D}_{PILE}$ with $10, 15, 20, ...$ blocks inside the bag in the situations $s_0(i)$. This would give the set

$$\{bagCount(10, s_0(1)), bagCount(15, s_0(2)), ...\} \cup \{pileCount(x, 0, s_0(i)) : x, i \in \mathbb{N}\}$$

as $\mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$; note that there are no negative literals here. Now one enumeration of $\mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$ might be:

$$\langle pileCount(2, 0, s_0(2)), \sharp, bagCount(10, s_0(1)), bagCount(15, s_0(2)), \sharp, ...\rangle$$

Recall that every element of $\mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$ must appear in an enumeration; so for any i, by examining a sufficiently large initial segment of an enumeration, we can be sure that all the initial fluent values for $s_0(i)$ have been specified.

The goal is specified in a logical formula; to fit in nicely with the query syntax, we specify a non-goal state. For Example 1, this is:

$$nongoal(s) \leftrightarrow \exists k_1, k_2, n_1, n_2 (pileCount(k_1, n_1, s) \wedge pileCount(k_2, n_2, s) \wedge k_1 \neq k_2 \wedge n_1 > 0 \wedge n_2 > 0 \wedge n_1 \neq n_2) \vee \exists k\, pileCount(k, 1, s)$$

Queries for Planning

Most deductive planning approaches (e.g. (De Giacomo *et al.* 1997; Levesque 1996)) answer an existential query to see if a (correct) plan exists, i.e.: is there a sequence of actions that, starting from the initial state, leads to a state where a given property (the goal) holds? In the situation calculus, using the domain described by $\mathcal{D}$ starting at situation s_0 (described by $\mathcal{D}_{s_0}$), this will be equivalent to discovering

$$\mathcal{D}_{s_0} \cup \mathcal{D} \cup \mathcal{P} \models \exists p, s_f\ moveto(p, s_0, s_f) \wedge goal(s_f) \quad (1)$$

and further whether it is possible to find a specific term as a witness for p. Conventional situation calculus approaches would also add second-order axioms to minimize the P predicate and restrict actions to standard terms.

If there is a planning language allowing for conditional plans and loops, then p, instead of being a sequence of actions, is a term in the planning language. In this case, directly solving planning problems using a naive theorem proving approach is difficult. For this reason, it has been proposed to solve this problem by solving a series of problem instances. The problem instances themselves should be demonstrably solvable, so were it possible to find a solution solving all of these problem instances, we will be able to produce a solution to the entire problem.

We choose the collection of initial situations for which we intend our correct plan to succeed by enumerating all the members of $\mathbb{D}$ true in some structure $\mathfrak{M}$.

Definition 4. *Say that there is a correct plan for some planning problem in the form* $\mathcal{D} \cup \mathcal{P} \cup \mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$[5] *if:*

$$\mathcal{D} \cup \mathcal{P} \cup \mathrm{Diag}_{\mathbb{D}}(\mathfrak{M}) \models^{\mathbb{D}} \exists p (plan(p) \wedge \forall i, s_f\ satisfiesGoal(p, s_0(i), s_f)) \quad (2)$$

The intended interpretation is that $plan(p)$ is true iff p is a member of $\mathbb{P}$, and $situation(x)$ is true iff is a member of $\mathbb{I}$. Define the abbreviation $satisfiesGoal$:

$$satisfiesGoal(p, s_0(i), s_f) \leftrightarrow situation(s_0(i)) \rightarrow \neg(moveto(p, s_0(i), s_f) \wedge nongoal(s_f))$$

Call the query in (2) the *problem query* for $\mathcal{D} \cup \mathcal{P} \cup \mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$. Note that the $satisfiesGoal$ abbreviation can be expressed as the negation of positive predicates; this is the requirement for a limiting definite query. This abbreviation corresponds to the subformula $\neg\chi(\overline{x}, \overline{y})$ in the LR query form above. Non-goals are used here as they happen to be more convenient to fit into the LR query syntax. It says: it never happens that a success initial situation leads to a failure situation, the desired safety property.

Note that s_f is in fact calculable from p and $s_0(i)$, so s_f could be expressed as a function of p and $s_0(i)$ were we not restricted to Herbrand models. The predicate above forms s_f in an analogous manner to PROLOG execution. The objective is to construct a witness for p. When implemented in the LR framework above and p-plan errors are found, the program backtracks to construct another potential plan. Should the execution of a particular p-plan be undecidable in the context of the action theory, execution will proceed until more data appears, and the p-plan generated on the back of the additional data is re-evaluated.

The $plan(p)$ is the $\psi(\overline{x})$ in the LR query.

Proposition 5. *Suppose* $\mathcal{D} \cup \mathcal{P} \cup \mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$ *is a limiting definite program describing a situation calculus domain (that is, interpretable in a situation calculus theory) with a finite number of fluents, which in any particular initial situation have at most a finite number of possible values. Then the LR procedure applied to query* (2) *will generate a successful witness iff the relation in* (2) *holds.*

Note that the LR procedure is the l-planner which in this case, will produce a plan which works in every initial situation. The proof is straightforward but tedious, and based on (Caldon & Martin 2004), which shows that a limiting definite program and query are solvable by LR iff a solution exists. The restriction to limiting definite programs is a substantial restriction. The finiteness criterion on the vocabulary is essential, as it permits one to avoid using the

[5] Recall $\mathcal{D} = \Sigma \cup \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{s_0}$ in conventional developments of the situation calculus.

Lloyd-Topor transformations directly, and express negation by constructing an alternative positive predicate representing the negated concept; this could also be done with a finite domain solver. The proof consists of verifying that the axiomatization of $\mathcal{P}$ can be expressed as a limiting definite program.

If we were to assume that the "generate" and "test" phases of the planning activity were decidable, then this strategy would also be trivial, but note that the paper (Lin & Levesque 1998) shows how to give a coding for a Turing machine with looping conditional plans.

Extensions

Comparison with KPLANNER

The KPLANNER system(Levesque 2005) can be viewed as a fragment of this procedure with an extra axiom added. The KPLANNER system works by distinguishing a particular fluent called the *planning parameter*, which is unknown at planning time and for which no loop would be required if the value were known. Whereas for KPLANNER it appears that every plan with a different planning parameter instance is expected to have a correct plan, here we demand the weaker condition that under these semantics, all members of $\mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$ must have a correct plan.

The KPLANNER system allows for the relatively efficient generation of plans with loops. Rather than enumerate all possible plans, it generates a conditional robot plan based on one particular fluent value (the *generate* value) using a regression-based technique and then attempts to see if there is an iterative robot plan equivalent to it, by *unwinding* the loop. Having built this plan, it then sets the planning parameter to the (much larger) *test* value, and checks that this robot plan works. Generation of the conditional plan is performed by regression in a similar system to INDIGOLOG. While the example theory is not regressible, it is not difficult to employ a KPLANNER style technique here to find a solution.

Suppose $\mathcal{D}_{s_0}$ are our initial conditions in a conventional situation calculus planning framework. To these we can add a fluent $parameter$, and any extra formulas required to further restrict the initial fluent values.

For instance, in the above example, we could set

$$parameter(n, s) \leftrightarrow bagCount(n, s),$$

so $parameter$ would correspond to the $bagCount$ fluent.

Note that while this approach will solve the KPLANNER examples, it is more general as we do not require the modal operator here. In our semantics, a correct plan may work for five initial blocks but not for four, whereas KPLANNER would interpret the fluent as meaning "there are (inclusively) up to five blocks on the table," encompassing the possibility of there being four blocks there.

Add the $kpGen$ and $kpTest$ predicate symbols to the vocabulary, and add the axioms:

$$kpGen(p, n) \leftrightarrow \exists s_f, i \;\; parameter(n, s_0(i)) \rightarrow moveto(p', s_0(i), s_f) \wedge goal(s_f) \wedge unwind(p, p')$$

$$kpTest(p, n) \leftrightarrow \exists s_f, i \;\; parameter(n, s_0(i)) \rightarrow moveto(p, s_0(i), s_f) \wedge goal(s_f)$$

If we were to be truer to the KPLANNER spirit, we would ensure that $moveto$ was regressible. Unwind converts the conditional robot plan into an iterative plan. This formula captures the KPLANNER 'generate' step. Let $\mathcal{P}'$ be $\mathcal{P}$ with this axiom and these $parameter$ formulas added.

Finding a p-plan which solves a problem instance (in a conventional situation calculus indexed by n_0) then amounts to solving for p: $\mathcal{D} \cup \mathcal{P}' \models \exists p \; kpGen(p, n_0)$.

In KPLANNER the following inference rule is being employed (abusing natural deduction style notation):

$$\frac{kpGen(p, s_0(n_{gen})) \quad kpTest(p, s_0(n_{test}))}{\forall n \in \mathbb{N} \; kpTest(p, s_0(n))}$$

Here n_{gen} is a generate value for a potential plan, and n_{test} is a test value for the potential plan. In KPLANNER, clever use of the modal situation calculus can permit a robot plan to be shown to work for numbers less than the generate or test value; we do not require this here. It is clear that this rule is unsound, and not difficult to find planning problems which encounter this unsoundness. For practical use of this system, an appropriate n_{test} must be found. A user may run KPLANNER several times with different choices for n_{test} in an attempt to invalidate their choice. The LR procedure explicitly incorporates this invalidation step, but does not depend on the decidability of the p-plan generation.

We can generalize the KPLANNER query and effectively borrow wholesale its plan generation technique by considering:

$$\mathcal{D} \cup \mathcal{P} \cup \mathrm{Diag}_{\mathbb{D}}(\mathfrak{M}) \models^{\mathbb{D}} \exists p(kpGen(p, n_{gen}) \wedge \forall i, s_f \, satisfiesGoal(p, s_0(i), s_f)) \tag{3}$$

In KPLANNER, the choice of $\mathfrak{M}$ is equivalent to choosing a mapping $\mu : \mathbb{N} \rightarrow 2^{\{f(v, s_0(v)) : v \in \mathcal{V}\}}$, where $\mathcal{V}$ is the collection of possible values of some particular fluent f, and all other fluents have the same value for all the $s_0(i)$. If we set $\mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$ to be $\{f(\mu(n), s_0(n)) : n \in \mathbb{N}\}$ for some particular μ, the correspondence between the frameworks becomes clear. In all the examples given in (Levesque 2005), μ is the identity mapping.

Other Limitations

It is not the case that this strategy will work for all problems of this style. The Σ_2 syntax (i.e. $\exists x \forall y \; matrix(x, y)$) of the limiting definite query is an essential and permits this strategy to work. For more complex queries, this condition breaks down; it is possible to output a particular p-plan cofintitely many times despite this being a non-working plan. Suppose we are interested in the following problem when we plan with exogenous actions which are presented to the planner as a finite sequence coded as the argument to some predicate in $\mathbb{D}$; sample members of $\mathbb{D}$ would be $exogenousAction([\alpha_1, ..., \alpha_k])$ for actions $\alpha_1, ..., \alpha_k$.

We don't give full details of how to build a situation calculus framework using exogenous actions here, but given these elements it is not difficult to imagine a framework. One can imagine adapting a fragment of the framework in the style of (Giacomo, Lesperance, & Levesque 1997) to attempt planning with induction and exogenous actions, and

using the sequence of actions provided in the execution of the robot program with exogenous actions. For this analysis a simpler framework suffices. The P predicate above is extended to give an additional argument, a sequence of potential exogenous actions. An exogenous action is performed in preference to the p-plan action unless the exogenous action is $\sharp$, representing a null action.[6]

Were such a framework constructed, one may wish to ask the following kinds of questions:

> Is there a correct plan p which works in some (possibly infinite) collection of initial situations $s(i)$ such that for all i where p causes the goal for i to be satisfied, there exists some sequence of exogenous actions e causing some additional goal to be reached.

We can formalize this as the query as:

$$\exists p\ plan(p) \wedge [\forall e, i\ (exogenousAction(e) \wedge situation(i)) \rightarrow satisfiesGoal(p, e, i) \wedge (\exists e'\ exogenousAction(e') \wedge satisfiesAdditionalGoal(p, e', i))] \tag{4}$$

Here the three-place $satisfiesGoal$ will be identical to the above binary $satisfiesGoal$ with the exception of a sequence of exogenous actions which may be provided for the planner to deal with. This assumes that $satisfiesAdditionalGoal$ is essentially identical to $satisfiesGoal$ but with a different predicate instead of $nongoal$ describing some additional goal. Constructing this is straightforward.

Unfortunately, it is not possible to construct a l-planner f which solves this problem. For suppose we have a Gödel coding scheme for the situation calculus language, and $\ulcorner x \urcorner$ is the number which codes for x. Suppose $\sigma_{i,e}$ is the sequence of the initial i elements of an enumeration e of codes of $\mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$. There is no learner f having the property that for all e $f(\sigma_{i,e}) = \ulcorner t_p \urcorner$ for some t_p which is a witness for p in (4) for cofinitely many i. It follows directly from this that there is no l-planner which converges iff (4) has a witness for p.

A formal proof would require showing that there are sets of Σ_3 complexity in the arithmetic hierarchy described by the above formula. Note that such a learner is constructible from the LR procedure for (2); this shows that there can be no analogous procedure to the LR procedure for the above problem.

If $exogenousAction$ were some non-recursive predicate a similar argument would apply. On the other hand if there is a decidable procedure allowing one to produce the appropriate exogenous action sequence from the p-plan and the initial situation, then this counter-example breaks down.

An Optimizing Planner

Suppose we instead want an optimal (correct) plan:

Example 6. *Solve the problem of Example 1 but with the additional condition that* **the piles be as large as possible**.

There is a clear similarity between this problem and finding the greatest common divisor of an infinite collection of numbers, and the infinite g.c.d. problem is a clear analogue to the lower-bound problem of the previous paper. The solution to this problem assumes an order on plans, with some plans being preferable to others. Suppose that our vocabulary contains the binary predicate symbol $\preceq$ for an order on plans.

Say that a partial order $(X, \preceq)$ has *finite tips* iff for all members x of X, the set of all $y \in X$ with $x \preceq y$ is finite. For example, the interpretation of $\preceq$ as $\leq$ over the negative numbers has finite tips, but this interpretation over the integers does not have finite tips. Let K be a collection of formulas such that for all Herbrand models $\mathfrak{M}$ of K, the interpretation of $\preceq$ in $\mathfrak{M}$ is fixed (independent of $\mathfrak{M}$) and is a partial order with finite tips.

Let φ be the closed formula $\exists \bar{x}(\psi_1 \wedge \forall \bar{y}(\neg\psi_2 \wedge \forall \bar{z}(\xi \rightarrow \neg\psi_3)))$ where:

- ψ_1, ψ_2 and ψ_3 are built from formulas, all of whose closed instances belong to $\mathbb{D}$ or theoretical atoms built from predicates of a limiting definite program, using $\wedge$ and $\vee$ only.
- ξ is built from formulas of the form $x \preceq z$, with x a variable that occurs in $\bar{x}$ and z a variable that occurs in $\bar{z}$, using $\vee$ and $\wedge$ only.

Then there exists a partial function f, defined on the set of finite sequences of members of $\mathbb{D}$ into $\{0, 1\}$ such that for all Herbrand models $\mathfrak{M}$ of K and for all enumerations e of $\mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$, f converges on e to 1 iff $\mathrm{Diag}_{\mathbb{D}}(\mathfrak{M}) \models \varphi$. Note that there is no guarantee that f is computable in general. But if K is defined by one of our extended logic programs, and if this logic program is able to generate the set of terms "better", w.r.t. $\preceq$, than a given closed term, then f can be made computable. This is implemented in an extension of LR, the *optimizing limiting resolution procedure*:

```
Set i = 0; Set k = 0
Loop:
   Set P = B ∪ {φ_1, ..., φ_i}
   Attempt query ψ_1 over P yielding {θ_1 ... θ_i}
   For each witness θ_j attempt ψ_2 · θ_j over P:
      If no success within i steps for j ≤ i
         choose the least such j
      Attempt query‡ ξ · θ_j producing θ'
         Attempt query ψ_3 · θ_j · θ'
            If succeeds, backtrack to ‡
            If does not succeed within
                i steps for j ≤ i
             choose least j and put θ_j in S_i
   Increment i; If S_i ≠ S_{i-1} increment k
   Report (S_i, k)
End Loop
```

Proposition 7. *Suppose $B \cup \mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$ is a planning problem, and $\preceq$ is finitely refutable[7] in the optimizing problem query for $B \cup \mathrm{Diag}_{\mathbb{D}}(\mathfrak{M})$. Then the optimizing LR procedure will have an upper bound for k iff the formula (5) is satisfied. If there is an upper bound, then the plan p_i produced by the optimizing LR procedure for this k will be a witness for p in (5).*

[6] See appendix for an axiomatization of this

[7] i.e. co-r.e.

Note that for given θ_i the collection of θ' is finite and decidable; each θ' will contain a witness for z. Now we can ask the query:

$$\mathcal{D} \cup \mathcal{P} \cup \mathrm{Diag}_{\mathbb{D}}(\mathfrak{M}) \models^{\mathbb{D}} \exists p\ kpGen(p, n_{gen}) \wedge \forall i, s_f\ satisfiesGoal(p, s_0(i), s_f) \wedge \forall p'\ [works(p', n_{gen}) \wedge \forall i', s'_f\ satisfiesGoal(p, s_0(i)', s'_f) \rightarrow \neg(p \prec p')\ ] \quad (5)$$

Recall $satisfiesGoal$ is the subformula in (2). This is similar to the (2) query with the additional restriction that the an optimal correct plan in the $\preceq$ ordering is chosen. The $\preceq$ predicate determines the number of stacks after p-plan execution with the n_{gen} value and performs reverse integer comparison on these values. Since this fluent is a natural number, it will have finite tips. Note that there are domains where many properties will not have finite tips.

Conclusion

This analysis gives insight into why the KPLANNER framework seems to work in practice, despite being unsound. When attempting to use KPLANNER for a task one chooses the test value so as to be likely to invalidate non-working (potential) plans—one plays devil's advocate, trying to find a way of destroying the constructed plan. The modal fluent semantics make it easy to test a finite range of values at once. What happens if the planner nevertheless produces an invalid plan? In practice the user of such a system would iterate a process of fiddling with this test value to invalidate the incorrect plan and eventually cause a working plan to be produced. The analysis of KPLANNER in this paper incorporates this "fiddle" process directly into the limiting planning semantics, and the resulting procedure is sound and complete. In this way one can argue that KPLANNER is a fragment of a sound and complete identification in the limit procedure, and this is the reason why the procedure works in practice. The procedure can then be extended to optimal planning. Unlike KPLANNER, instead of the task being: "given a goal, find a robot program that achieves it", the task is transformed into: given a goal and a collection of initial conditions, find a robot program which describes those initial conditions. A robot program is now a pattern that describes precisely the set of initial conditions of interest.

Clearly, an alternative to this enumeration is to provide some finitely describable property which characterizes the success initial situations; however, for some problems, such a description may not exist or may not be easily available. In particular, the case (as here) where the domain is potentially infinite requires a different approach to that employed in conventional planning, and extends existing work on planning with incomplete information. One could argue that the semantics are unrealistic, inasmuch as the soundness and completeness are limiting properties, and thus any planner will not be able to detect whether a plan is successful or not. We have shown that the limiting property is essential to a subclass of these problems, and that it is *impossible* to attain better results in general. Nevertheless, we have shown that it is still possible to produce workable inductive planners despite these constraints.

It is not straightforward whether there is a difference in power between the simple limiting planner and the optimizing planner; that is, whether for any optimizing (program,query) pair, there is a simple (problem, query) pair such that the solution to the simple problem is also a solution to the optimizing problem for any choice of structure to be enumerated. We conjecture that there is a difference in the strength between the two planners.

References

Alchourron, C.; Gardenfors, P.; and Makinson, D. 1985. On the logic of theory change: Partial meet contraction and revision functions. *The Journal of Symbolic Logic* 50(2):510–530.

Baral, C., and Gelfond, M. 1999. Reasoning agents in dynamic domains. In Minker, J., ed., *Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14–16, 1999*. College Park, Maryland: Computer Science Department, University of Maryland.

Caldon, P., and Martin, E. 2004. Limiting Resolution: from theory to implementation. In *Proceedings of 20th Internnational Conference on Logic Programming*. St Malo, France: Springer Verlag.

De Giacomo, G.; Iocchi, L.; Nardi, D.; and Rosati, R. 1997. Planning with sensing for a mobile robot. In *Proceedings of the European Conference on Planning (ECP-97)*, number 1348 in Lecture Notes in Artificial Intelligence, 156–168. Springer-Verlag.

Giacomo, G. D.; Lesperance, Y.; and Levesque, H. J. 1997. Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In *IJCAI*, 1221–1226.

Gold, E. M. 1967. Language identification in the limit. *Information and Control* 10.

Jain, S.; Osherson, D.; Royer, J.; and Sharma, A. 1999. *Systems that Learn*. Cambridge, Mass., USA: M.I.T. Press, second edition.

Levesque, H. 1996. What is planning in the presence of sensing? In *Proceedings of AAAI-96 Conference*, 1139–1146.

Levesque, H. 2005. Planning with loops. In *Proceedings of the IJCAI-05 Conference*.

Lin, F., and Levesque, H. J. 1998. What robots can do: Robot programs and effective achievability. *Artificial Intelligence* 101(1-2):201–226.

Martin, E.; Sharma, A.; and Stephan, F. 2002. Logic, learning and topology in a common framework. In Cesa-Bianchi, N.; Numao, M.; and Reischuk, R., eds., *Proceedings of the 13th International Conference on Algorithmic Learning Theory*. Springer Verlag.

Reiter, R. 2001. *Knowledge in Action*. MIT Press.