Using SAT and Logic Programming to Design Polynomial-Time Algorithms for Planning in Non-deterministic Domains

Chitta Baral
Computer Science and Engineering
Arizona State University
Tempe, AZ 85287, USA
chitta@asu.edu

Thomas Eiter
Institute of Information Systems
Faculty of Informatics
Vienna University of Technology
A-1040 Vienna, Austria
eiter@kr.tuwien.ac.at

Jicheng Zhao
Computer Science and Engineering
Arizona State University
Tempe, AZ 85287, USA
jicheng@asu.edu

Abstract

We show that a Horn SAT and logic programming approach to obtain polynomial time algorithms for problem solving can be fruitfully applied to finding plans for various kinds of goals in a non-deterministic domain. We particularly focus on finding weak, strong, and strong cyclic plans for planning problems, as they are the most studied ones in the literature. We describe new algorithms for these problems and show how non-monotonic logic programming can be used to declaratively compute strong cyclic plans. As a further benefit, preferred plans among alternative candidate plans may be singled out this way. We give complexity results for weak, strong, and strong cyclic planning. Finally, we briefly discuss some of the kinds of goals in non-deterministic domains for which the approach in the paper can be used.

Introduction and Motivation

In recent years, one of the approaches that has been used in finding solutions to AI problems is to find "models" of a logical encoding of the problem. Examples of this include finding planning via satisfiability encoding (Kautz & Selman 1992) or logic programming encodings with answer set semantics (Gelfond & Lifschitz 1991). The later is now referred to as answer set programming. But in most of these cases, the problems that are solved are in the complexity class NP-complete or beyond. One outlier is the work (Baral & Eiter 2004) which takes advantage of the lower complexity results about specific logic programming and SAT subclasses to come up with a polynomial-time algorithm for finding maintenance policies.

In that paper, the authors first give a propositional SAT encoding of the problem. They then give a transformation of that encoding to a propositional reverse Horn encoding and show that the models of the encoding correspond to desired agent policies. The fixpoint iteration approach to compute models of Horn theories, which is feasible in linear time, is then exploited to develop a genuine polynomial-time algorithm for finding agent policies. If one were to view the logical encoding as a specification, then the above mentioned approach can be considered as a systematic way to develop algorithms from specifications.

In this paper, we show that the above approach can also be used in finding polynomial time algorithms for some planning problems that emerge in non-deterministic domains (Dal Lago, Pistore, & Traverso 2002; Cimatti *et al.* 2003). In particular, we consider the notions of strong planning, weak planning, and strong cyclic planning (Cimatti *et al.* 2003) and develop encodings inspired by the encoding in (Baral & Eiter 2004), leading to polynomial time algorithms for finding strong cyclic plans, strong plans and weak plans. Overall, our main contributions in this paper are:

- We illustrate the novel algorithm design approach of (Baral & Eiter 2004) to systematically develop an algorithm from a logical specification. Passing through Horn SAT specifications, we develop new polynomial time algorithms for weak, strong, and strong cyclic planning; thus shedding additional insights about these notions.
- We show how strong cyclic plans can be declaratively generated with answer set programming at an abstract level.
- We discuss how particular properties of the encodings and features of answer set solvers can be exploited for computing (most) preferred plans among alternative candidate plans. In particular, based on the encoding, maximal plans and least defined plans can be found in polynomial time, and in an answer set solver certain preference information (e.g. based on action cost) can be easily expressed.
- We briefly discuss how our approach can lead to algorithms for certain other kind of goals in non-deterministic domains, such as maintenance goals and goals composed of a sequence of achievements.
- We give complexity results about weak, strong and strong cyclic planning. (No such results appear in previous papers.)

Plans and Goals in Non-deterministic Domains

We start with recalling the notions of weak, strong, and strong cyclic plans from (Cimatti *et al.* 2003). Such plans manifest in non-deterministic domains. In such domains, plans map states to actions or to sets of actions. A weak plan to achieve p is a plan that says that at least one of the paths (based on following that plan) leads to p. A strong plan to achieve p is a plan that says that all paths (based on following that plan) would lead to p. A strong cyclic plan to achieve p is a plan that says all along the path (based on following that plan) there is at least one of the paths (by following that

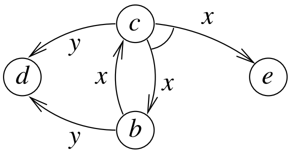

Figure 1: Transition diagram of the planning domain $\mathcal{D}$

plan) that would lead to p. In the language of π-CTL* (Baral and Zhao 2004), these goals are expressed as $\mathsf{E}_\pi\Diamond p$, $\mathsf{A}_\pi\Diamond p$, and $\mathsf{A}_\pi\Box(\mathsf{E}_\pi\Diamond p)$, respectively; where $\Diamond$ means eventually, $\Box$ means always, E_π means exists a path following the plan under consideration, and A_π means all paths following the plan under consideration.

We now give the formal definitions.

Definition 1 (Planning Domain) *A Planning Domain is a triple* $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Phi \rangle$ *where*

- $\mathcal{S}$ *is the set of states,*
- $\mathcal{A}$ *is the finite set of actions,*
- $\Phi : \mathcal{S} \times \mathcal{A} \rightarrow 2^{\mathcal{S}}$ *is the (non-deterministic) transition function that specifies how the state of the world changes in response to actions.*

Definition 2 (Executable Actions) *Given a planning domain* $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Phi \rangle$*, for each* $s \in \mathcal{S}$*, the set of executable actions in s,* $Act(s)$*, is* $Act(s) = \{a : \Phi(s, a) \neq \emptyset\}$.

Example 1 *Consider a planning domain* $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Phi \rangle$. *Let* $\mathcal{S} = \{b, c, d, e\}$, $\mathcal{A} = \{x, y\}$*, and the transition function* Φ *as in Figure 1. Then,* $Act(b) = \{x, y\}$ *while* $Act(e) = \emptyset$.

Definition 3 (Plan) *Given a planning domain* $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Phi \rangle$*, a mapping* $\pi : \mathcal{S} \rightarrow 2^{\mathcal{A}}$ *is called a plan if for every* $s \in \mathcal{S}$*, if* $\pi(s)$ *is defined, then* $\pi(s) \subseteq Act(s)$.

Definition 4 (Planning Problem) *Let* $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Phi \rangle$ *be a planning domain. A planning problem for* $\mathcal{D}$ *is a triple* $\langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$ *where* $\mathcal{I} \subseteq \mathcal{S}$*, and* $\mathcal{G} \subseteq \mathcal{S}$.

Definition 5 (Execution Structure) *Let* π *be a plan of a planning domain* $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Phi \rangle$*. The execution structure induced by* π *from the set of initial states* $\mathcal{I} \subseteq \mathcal{S}$ *is a tuple* $K = \langle Q, T \rangle$ *with* $Q \subseteq \mathcal{S}$ *and* $T \subseteq \mathcal{S} \times \mathcal{S}$ *inductively defined as follows:*

1. *If* $s \in \mathcal{I}$*, then* $s \in Q$*, and*
2. *If* $s \in Q$*, action* $a \in \pi(s)$*, and* $s' \in \Phi(s, a)$*, then* $s' \in Q$ *and* $(s, s') \in T$.

A state $s \in Q$ *is a terminal state of* K *if there is no* $s' \in Q$ *such that* $(s, s') \in T$.

In the following, given a $\langle Q, T \rangle$, we say a state $s_2 \in Q$ is reachable from state $s_1 \in Q$ if there is a path from s_1 to s_2 in T.

Definition 6 (Plans with respect to a planning problem) *Let* $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Phi \rangle$ *be a planning domain,* $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$ *be a planning problem,* π *be a plan in* $\mathcal{D}$*. Let* $K = \langle Q, T \rangle$ *be the execution structure induced by* π *from* $\mathcal{I}$.

1. π *is a weak plan with respect to* P *iff for any state in* $\mathcal{I}$*, some terminal state in* $\mathcal{G}$ *is reachable from the state.*
2. π *is a strong plan with respect to* P *iff* K *is acyclic and all the terminal states of* K *are in* $\mathcal{G}$.
3. π *is a strong cyclic plan with respect to* P *iff from any state in* Q *some terminal state is reachable and all the terminal states of* K *are in* $\mathcal{G}$.

Example 2 *Continuing Ex. 1, for the planning problem* $\langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$ *where* $\mathcal{I} = \{b\}$ *and* $\mathcal{G} = \{e\}$*, the mapping* π *such that* $\pi(c) = x$ *and* $\pi(b) = x$*, is a strong cyclic plan. Its execution structure is* $K = \{\{b, c, e\}, \{(b, c), (c, b), (c, e)\}\}$*. In this planning problem, no strong plan exists, while* π *is also a weak plan.*

Finding Strong Cyclic Plans

In this section we use the approach in (Baral & Eiter 2004) to develop algorithms that construct strong cyclic plans. To start with, we give a propositional SAT encoding of a planning problem, and show that the models of this theory encode strong cyclic plans, if one exists, and vice versa.

SAT encoding S-$Cyclic(P)$

In our SAT encoding, we will use, for each state s and action a, propositions s_i and s_a_i, where $i \geq 0$ is an integer. Intuitively, s_i will mean that there is a path from s to $\mathcal{G}$, following T of the execution structure $K = \langle Q, T \rangle$, of length at most i. Similarly, s_a_i will intuitively mean that there is a path from s to $\mathcal{G}$, following T of the execution structure $K = \langle Q, T \rangle$, of length at most i, with a as its first action. We employ an upper bound max for i, depending on the number of states in $\mathcal{S}$; if there is no path of length at most max, there is no path at all.

Algorithm 1 *Suppose we are given a planning problem* $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$ *where* $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Phi \rangle$*. Let* $max = |\mathcal{S}| - 1$*. We translate* P *into a SAT encoding* S-$Cyclic(P)$ *as follows:*

(0) for all $s \in \mathcal{S}$ *and* i, $0 < i \leq max$: $s_{i-1} \Rightarrow s_i$

(1) for every state $s \in \mathcal{S} \backslash \mathcal{G}$*, and for all* i, $0 < i \leq max$: $s_i \Rightarrow \bigvee_{a \in Act(s)} s\_a_i$

(2) for every states s, $s' \in \mathcal{S}$ *such that* $s' \in \Phi(s, a)$ *for some action* a: $s\_a_{max} \Rightarrow s'_{max}$

(3) for every state $s \in \mathcal{S}$*, action* $a \in Act(s)$*, and for all* i, $0 < i \leq max$: $s\_a_i \Rightarrow \bigvee_{s' \in \Phi(s,a)} s'_{i-1}$

(4) for every state $s \in \mathcal{S}$*, action* $a \in Act(s)$*, and* $1 < i \leq max$: $s\_a_{i-1} \Rightarrow s\_a_i$

(5) for $s \in \mathcal{I}$: s_{max}

(6) for $s \in \mathcal{S} \setminus \mathcal{G}$: $\neg s_0$

The intuition behind this encoding is as follows. The clauses in (0) state that if there is a path from s to $\mathcal{G}$ of length at most i-1, then there is a path of length at most i. The clauses in (4) make a similar statement for paths with first action a. The clauses in (1) state that if there is a path from s to $\mathcal{G}$ of length at most i, then there must exist an action a which is the first action of such a path. The clauses in (2) state that for any state s, there is a path from s to $\mathcal{G}$ of length at most max with a as its first action only if from every state $s' \in \Phi(s, a)$ a path to $\mathcal{G}$ of length at most max exists. This takes into account the possibility that s may be in (the closure) Q of the execution structure $\langle Q, T\rangle$. The clauses in (3) state that a path from s to $\mathcal{G}$ of length at most i with a as its first action exists only if there is a path from some state $s' \in \Phi(s,a)$ to $\mathcal{G}$ of length at most i-1. The clauses in (5) state that every initial state must have a path of length at most max. Finally, the clauses in (6) exclude paths of length zero for non-goal states.

Strong cyclic plans with respect to P and the models of $S\text{-}Cyclic(P)$ are formally connected as follows.

Proposition 1 *Let $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G}\rangle$ be a planning problem with planning domin $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Phi\rangle$.*

1. *P has a strong cyclic plan iff $S\text{-}Cyclic(P)$ is satisfiable;*
2. *For any model M of $S\text{-}Cyclic(P)$, the partial function $\pi_M : \mathcal{S} \to 2^{\mathcal{A}}$ defined by $\pi_M(s) = \{a \mid M \models s\_a_j, j = \min_i M \models s_i\}$ on all states $s \in \mathcal{S} \setminus \mathcal{G}$ such that $M \models s_i$ for some i, is a strong cyclic plan of P.*

Horn SAT Encoding

While $S\text{-}Cyclic(P)$ is constructible in polynomial time from P, we can not automatically infer that finding strong cyclic plans is polynomial, since SAT is a canonical NP-hard problem. However, a closer look at the structure of the clauses in $S\text{-}Cyclic(P)$ reveals that this instance is solvable in polynomial time. Indeed, it is a *reverse Horn* theory; i.e., after reversing the propositions, the theory is Horn. Using propositions $\overline{s_i}$, which intuitively mean the converse of s_i, the Horn theory corresponding to $S\text{-}Cyclic(P)$, denoted $\overline{S\text{-}Cyclic}(P)$, is as follows:

Algorithm 2 *We are given a planning problem $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G}\rangle$, where $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Phi\rangle$. Suppose $max = |\mathcal{S}|-1$. We translate P into a Horn encoding $\overline{S\text{-}Cyclic}(P)$:*

(0) for all $s \in \mathcal{S}$ and i, $0 < i \leq max$: $\overline{s_i} \Rightarrow \overline{s_{i-1}}$

(1) for every state $s \in \mathcal{S}\backslash\mathcal{G}$, and for all i, $0 < i \leq max$: $\bigwedge_{a \in Act(s)} \overline{s\_a_i} \Rightarrow \overline{s_i}$.

(2) for every states s, $s' \in \mathcal{S}$ such that $s' \in \Phi(s,a)$ for some action a: $\overline{s'_{max}} \Rightarrow \overline{s\_a_{max}}$

(3) for every state $s \in \mathcal{S}$, action $a \in Act(s)$, and for all i, $0 < i \leq max$: $\bigwedge_{s' \in \Phi(s,a)} \overline{s'_{i-1}} \Rightarrow \overline{s\_a_i}$

(4) for every state $s \in \mathcal{S}$, action $a \in Act(s)$, and for all i, $1 < i \leq max$: $\overline{s\_a_i} \Rightarrow \overline{s\_a_{i-1}}$

(5) for $s \in \mathcal{I}$: $\overline{s_{max}} \Rightarrow \perp$

(6) for $s \in \mathcal{S} \setminus \mathcal{G}$: $\overline{s_0}$

As computing a model of a Horn theory is a well-known polynomial problem (Dowling & Gallier 1984), we thus obtain the following result.

Theorem 1 Strong cyclic plans can be computed in polynomial time. $\square$

Maximal plan An interesting aspect of the above is that, as well-known, each satisfiable Horn theory T has the least model, $M^*(T)$, which is given by the intersection of all its models. Moreover, the least model is computable in linear time, cf. (Dowling & Gallier 1984). This model not only leads to a strong cyclic plan, but also leads to the *maximal* plan, in the sense that the control is defined on the greatest set of states outside $\mathcal{G}$ among all possible strong cyclic plans for initial states $\mathcal{I}'$ and goal states $\mathcal{G}$ such that $\mathcal{I} \subseteq \mathcal{I}'$. This gives a clear picture of which other states may be added to $\mathcal{I}$ while strong cyclicness is preserved. Besides, for any strong cyclic plan, for any state, the action dictated by that plan for that state is among the actions dictated by the plan corresponding to $M^*(T)$.

Lean plans On the other hand, intuitively a strong cyclic plan constructed from some maximal model of $\overline{S\text{-}Cyclic}(P)$ with respect to the propositions $\overline{s_k}$ is undefined to a largest extent, and works merely for a smallest extension. We may generate, starting from any model of T, such a maximal model of T by trying to flip step by step all propositions $\overline{s_k}$ which are $false$ to $true$, and change other propositions as needed for satisfiability. In this way, we can generate a maximal model of T on $\{\overline{s_k} \mid s \in \mathcal{S} \setminus E\}$ in polynomial time, from which a "lean" control can also be extracted in polynomial time.

Genuine Procedural Algorithm

From the encoding to Horn SAT above, we can distill a direct algorithm STRONG CYCLIC PLAN to construct a strong cyclic plan, if one exists. It mimics the steps which a SAT solver might take in order to solve $\overline{S\text{-}Cyclic(P)}$. For each state $s \in \mathcal{S}$ and action $a \in Act(s)$, we use counters $c[s]$ and $c[s\_a]$ ranging over $\{-1, 0, \cdots, max\}$ and $\{0, 1, \cdots, max\}$, respectively. Intuitively, $c[s] = i$ represents that so far $\overline{s_0}$, $\overline{s_1}$, $\cdots$, $\overline{s_i}$ are assigned true; in particular, $i = -1$ represents that no $\overline{s_i}$ is assigned true yet. Similarly, $c[s\_a] = i$ represents that so far $\overline{s\_a_1}$, $\overline{s\_a_2}$, $\cdots$, $\overline{s\_a_i}$ are assigned true. In particular, $c[s\_a_i] = 0$ means that no $\overline{s\_a_i}$ is assigned true yet.

Algorithm 3 STRONG CYCLIC PLAN

Input: *A planning domain $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Phi\rangle$, and a planning problem $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G}\rangle$.*

Output: *A strong cyclic plan of P if such plan exists. Otherwise, output that no such plan exists.*

(Step 1) Initialization:

(i) For every $s \in \mathcal{G}$, set $c[s] := -1$.

(ii) For every $s \in \mathcal{S}\backslash\mathcal{G}$, if $Act(s) = \emptyset$ then set $c[s] := max$ else set $c[s] := 0$.

(iii) For each $s \in \mathcal{S}\backslash\mathcal{G}$ and $a \in Act(s)$, set $c[s\_a]$:=0.

(Step 2) Repeat until no change or $c[s] = max$ for some $s \in \mathcal{I}$:

(i) For every state $s \in \mathcal{S}\backslash\mathcal{G}$ such that $Act(s) \neq \emptyset$, $c[s] := max(c[s], i)$ where $i = \min_{a \in Act(s)} c[s\_a]$.

(ii) For every state $s \in \mathcal{S}$, $a \in Act(s)$, and $s' \in \Phi(s,a)$, if $c[s'] = max$, then $c[s\_a] := max$.

(iii) For every state $s \in \mathcal{S}$ and $a \in Act(s)$, $c[s\_a] := max(c[s\_a], i{+}1)$ where $i{=}\min_{s' \in \Phi(s,a)} c[s']$.

(Step 3) If $c[s] = max$ for some $s \in \mathcal{I}$, then output that there is no strong cyclic plan; halt.

(Step 4) Output the plan $\pi : \mathcal{S} \to 2^{\mathcal{A}}$ defined on the states $s \in \mathcal{S}\backslash\mathcal{G}$ with $c[s] \leq max$ and $\pi(s) = \{a \mid a \in Act(s), c[s\_a] = \min_{b \in Act(s)} c[s\_b]\}$. $\square$

Proposition 2 *Algorithm* STRONG CYCLIC PLAN *finds a strong cyclic plan, if one exists, in a planning problem. Furthermore, for every input $\mathcal{D}$ and P, it terminates in polynomial time.*

We remark that algorithm STRONG CYCLIC PLAN can be made more efficient by pruning in a linear time preprocessing all states which are not on a path between some states $s \in \mathcal{I}$ and $s' \in \mathcal{G}$.

A more detailed account of the complexity of STRONG CYCLIC PLAN and possible improvements are given below.

Strong Cyclic Planning Using an Answer Set Solver

In this section, we show how computing strong cyclic plans can be encoded as a logic program, based on the results of the previous section. More precisely, we describe an encoding to non-monotonic logic programs under the Answer Set semantics (Gelfond & Lifschitz 1991), which can be executed on one of the available Answer Set solvers such as DLV (Leone *et al.* 2005) or Smodels (Simons, Niemelä, & Soininen 2002). These solvers support the computation of answer sets (models) of a given program, from which solutions (in our case, strong cyclic plans) can be extracted.

The encoding is generic, i.e., given by a *fixed program* which is evaluated over instances I represented by input facts $F(I)$. It makes use of the fact that non-monotonic logic programs can have multiple models, which correspond to different solutions, i.e., different strong cyclic plans.

In the following, we first describe how a system is represented in a logic program, and then we develop the logic programs for both deterministic and general, nondeterministic domains. We adopt here the syntax of DLV; the adaptions for other Answer Set Solvers (e.g. Smodels) are very minor.

Input representation $F(I)$ The input I can be represented by facts $F(I)$ as follows.

- The following facts represent the planning domain $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Phi \rangle$ and the planning problem $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$:
- `state(`s`)`, for each $s \in \mathcal{S}$;
- `action(`a`)`, for each $a \in \mathcal{A}$;
- `trans(`s, a, s'`)`, for each $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$ such that $s' \in \Phi(s, a)$;
- the set of states $\mathcal{I}$ is represented by using a predicate `start` by facts `start(`s`)`, for each $s \in \mathcal{I}$;
- the set of states $\mathcal{G}$ is represented by using a predicate `goals` by facts `goal(`s`)`, for each $s \in \mathcal{G}$;
- finally, the ranges $1 \ldots max$ and $2 \ldots max$ are represented using predicates $range1$ and $range2$, respectively.

Program P_{SC} The program P_{SC}, executable on the DLV engine, for computing a strong cyclic plan is as follows.

```
%ranges
 range1(N) :- #int(N), N>0.
 range2(N) :- #int(N), N>1.
% 0
 s_bar(S,J1) :-  s_bar(S,J), J=J1+1.
% 1
 s_bar(S,I) :- state(S), not goal(S),
          range1(I), not fail_body(S,I).
 fail_body(S,I) :- range1(I),
          trans(S,A,Y), not s_a_bar(S,A,I).
% 2
 s_a_bar(S,A,#maxint) :-
          trans(S,A,Y), s_bar(Y,#maxint).
% 3
 s_a_bar(S,A,I) :- state(S), action(A),
          range1(I), not fail_a_body(S,A,I).
 fail_a_body(S,A,I) :- range1(I), I=I1+1,
          trans(S,A,Y), not s_bar(Y,I1).
% 4
 s_a_bar(S,A,I1) :- range2(I),
          I=I1+1, s_a_bar(S,A,I).
% 5
 :- s_bar(S,#maxint), start(S).
% 6
 s_bar(S,0) :- state(S), not goal(S).
% single out a plan:
  pi(S,A) :- fail_a_body(S,A,J), not goal(S),
             J = #min{J1: fail_body(S,J1)}.
```

Besides the input predicates of $F(I)$, the program employs predicates `s_bar(S,I)` and `s_a_bar(S,A,I)` which intuitively correspond to $\overline{S_I}$ and $\overline{S\_A_I}$ respectively. The predicates `fail_body(S,I)` and `fail_a_body(S,A,I)` are used to uniformly represent clauses in (1) and (3), respectively, with varying body size; they amount to the negation of `s_bar(S,I)` and `s_a_bar(S,A,I)`, respectively. The plan is computed in the predicate `pi(S,A)`.

Example 3 *The logic program encoding $F(I)$ of the strong cyclic planning problem in Example 2 is as follows:*

```
#maxint=3.
state(b). state(c). state(d). state(e).
start(b).  goal(e). action(x). action(y).
trans(b,x,c). trans(c,x,b). trans(c,x,e).
trans(b,y,d). trans(c,y,d).
```

The program $P_{SC} \cup F(I)$ has one answer set. Filtered to the atoms `fail_a_body(s,a,i)` *and* `pi(s,a)`*, we get:*

```
{ fail_a_body(c,x,1), fail_a_body(b,x,2),
  fail_a_body(c,x,2), fail_a_body(b,x,3),
  fail_a_body(c,x,3), pi(b,x), pi(c,x)   }
```

Hence, we obtain the strong cyclic plan π given by $\pi(b) = \{x\}$ and $\pi(c) = \{x\}$.

Preferred plans In the above example, P_{SC} yields a single and deterministic plan π, i.e., $|\pi(s)| \leq 1$ always holds. In general, there can be multiple answer sets, each corresponding to a different plan. Moreover, π can be nondeterministic; if in Example 3 a further action z would lead from c to e, then $\pi(c, e)$ would be in the result computed, and thus $\pi(c) = \{x, z\}$. By adding further rules in P_{SC}, we can easily generate a deterministic plan π_{det}, e.g. by nondeterministically selecting one action from $\pi(s)$:

```
pi_det(S,A) :- pi(S,A), not drop(S,A).
drop(S,A) v drop(S,B):- pi(S,A), pi(S,B), A<>B.
```

For the case where multiple solutions exist, we might exploit features available in Answer Set Solvers to select preferred plans. For example, using weak constraints offered by DLV, we can express prioritization between different actions. For illustration, the weak constraints

```
:~ pi_det(c,x). [:1]    :~ pi_det(c,z). [:2]
```

express that as for π_{det}, taking action z in state c is preferred over taking x. Using weak constraints, we can also easily model *costs* for action execution, possibly dependent on the state, which add up in execution. In this way, optimal (i.e., most preferred) plans among the candidates can be computed, possibly combining different criteria like deterministic actions and execution cost. We leave a detailed discussion of this for further study.

Finding Strong Plans

Finding strong plans can be approached in three ways: (i) as a special case of finite maintainability, when there are no exogenous actions; (ii) further constraining strong cyclic planning; or (iii) by a generic SAT encoding.

As for (ii), a Horn SAT encoding and genuine algorithm for strong planning are as follows:

Horn Sat Encoding $\overline{Strong}(P)$: The clauses (0), (1), (4), (5), (6) from $\overline{S\text{-}Cyclic}(P)$ and the following clauses:

(7) For every state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, for all $s' \in \Phi(s,a)$, and for all i, $0 < i \leq max$: $\overline{s'_{i-1}} \Rightarrow \overline{s\_a_i}$

Genuine procedure STRONG PLAN**:** Steps 1, 2.(i), 3, and 4 from STRONG CYCLIC PLAN plus the new Step:

(Step 2) (ii$'$) For any state $s \in \mathcal{S}$, if $s' \in \Phi(s,a)$ for $a \in Act(s)$ and $c[s'] = i$ such that $0 \leq i \leq max$, then do $c[s\_a] := max(c[s\_a], i+1)$.

As discussed later, this yields algorithms of the same order as for strong cyclic planning.

The following Horn SAT encoding and the corresponding genuine procedure is more efficient.

Algorithm 4 *For planning problem $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Phi \rangle$, the Horn instance $\overline{Strong}^+(P)$ contains:*

(0) for every $s \in \mathcal{G}$: s

(1) for every state $s \in \mathcal{S} \backslash \mathcal{G}$ and action $a \in Act(s)$ such that $\Phi(s,a) = \{s'_1, \ldots, s'_m\}$, $m > 0$:
$s'_1 \wedge \cdots \wedge s'_m \Rightarrow s$ and $s'_1 \wedge \cdots \wedge s'_m \Rightarrow s\_a$.

(2) For $\mathcal{I} = \{s_1, \ldots, s_l\}$: $s_1 \wedge \cdots \wedge s_l \Rightarrow \bot$.

Theorem 2 *For a planning problem $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$,*

(i) *a strong solution exists iff $\overline{Strong}^+(P)$ is unsatisfiable iff $\bot$ is derivable from $\overline{Strong}^+(P)$.*

(ii) *$\pi = \{\langle s, a \rangle \mid s\_a \in T^i_{P'}, s \notin T^{i-1}_{P'}$, for some $i \geq 1\}$, is a (non-deterministic) strong solution, where $T^1_{P'} = \mathcal{G}$ and $T^{i+1}_{P'} = \{\ell \mid \ell_1 \wedge \cdots \wedge \ell_l \Rightarrow \ell \in \overline{Strong}^+(P)$ and $\ell_1, \ldots, \ell_l, \in T^i_{P'}\}$ for $i \geq 1$, are the powers $T^i_{P'}$ of the logic programming immediate consequence operator $T_{P'}$ (see e.g. (Dantsin* et al. *2001)) for the program $P' = \overline{Strong}^+(P)$ (viewing $\bot$ as atom).*

A strong plan π as in the theorem can be constructed in $O(|\Phi| + |\mathcal{S}|)$ time starting from P, since $\overline{Strong}^+(P)$ is easily constructed and, as well-known, the powers of $T_{P'}$ are incrementally computable in linear time using proper data structures, cf. remarks in (Dantsin *et al.* 2001).

Finding Weak Plans

One way to think about finding weak plans is as relaxing strong cyclic planning. A respective Horn SAT encoding and genuine algorithm for Weak planning are as follows:

Horn Sat Encoding $\overline{Weak}(P)$: The clauses (0), (1), (3), (4), (5), (6) from $\overline{S\text{-}Cyclic}(P)$.

Genuine procedure: It consists of Steps 1, 2.(i), 2.(iii), 3, and 4 of algorithm STRONG CYCLIC PLAN. (It does not contain the Step 2 (ii).)

Again, this yields algorithms of the same order as for strong cyclic planning. More efficient ones emerge from the following encoding.

Algorithm 5 *For planning problem $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Phi \rangle$, the Horn instance $\overline{Weak}^+(P)$ is as follows:*

(0) for every $s \in \mathcal{G}$: s

(1) for every state $s \in \mathcal{S} \backslash \mathcal{G}$, action $a \in Act(s)$, and $s' \in \Phi(s,a)$: $s' \Rightarrow s\_a$ and $s' \Rightarrow s$.

Theorem 3 *For a planning problem $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$,*

(i) *a weak solution exists iff for each $s \in \mathcal{I}$, $\overline{Weak}^+(P) \cup \{\neg s\}$ is unsatisfiable if and only if each $s \in \mathcal{I}$ is true in $M^*(\overline{Weak}^+(P))$, the least model of $\overline{Weak}^+(P)$.*

(ii) *$\pi = \{\langle s, a \rangle \mid s\_a \in M^*(\overline{Weak}^+(P))\}$, is a (non-deterministic) strong solution, if any strong solution exists.*

Note that $\overline{Weak}^+(P)$ is definite Horn, and thus its least model $M^*(\overline{Weak}^+(P))$ does exist. Furthermore, it is computable in linear time in the size of $\overline{Weak}^+(P)$. Since the latter is easily constructed, finding a weak plan w.r.t. P is thus feasible in time $O(|\Phi| + |\mathcal{S}|)$, i.e., in linear time.

Complexity and Relation to other Algorithms

For any planning domain $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \Phi \rangle$ and planning problem $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$, we denote by $\|\mathcal{D}\| = |\mathcal{S}| + |\mathcal{A}| + |\Phi|$ and $\|P\| = \|\mathcal{D}\| + |\mathcal{I}| + |\mathcal{G}|$ the representation size of $\mathcal{D}$ and P, respectively (where Φ is viewed as set of triples $\langle s, a, s' \rangle$).

Proposition 3 *Strong Cyclic Planning can be solved, via the Horn encoding $\overline{S\text{-}Cyclic}(P)$ and, by a suitable implementation of Algorithm* STRONG CYCLIC PLAN, *in time $O(|\mathcal{S}| \cdot \|P\|)$ and $O(|\mathcal{S}| \cdot |\Phi|)$, respectively.*

Compared to (Cimatti *et al.* 2003), our algorithm for strong cyclic planning works differently. Basically, their algorithm iteratively computes weak plans by backtracking from the goal states and prunes the planning problem until a weak plan which is also a strong cyclic plan is obtained. Our algorithm, instead, has no such intuition and simply aims at establishing the necessary logical conditions, as in the seminal planning as satisfiability approach (Kautz & Selman 1992). A simple implementation of the Cimatti *et al.* algorithm has $O(|\mathcal{S}|^2|\Phi|)$ time complexity, while a sophisticated one has $O(|\mathcal{S}| \cdot |\Phi|)$ comparable to ours. In the extended version of the paper, we illustrate on an example the difference between the workings of their algorithm and ours.

For finding strong plans and weak plans by constrained and relaxed strong cyclic planning, respectively, we obtain:

Proposition 4 *Strong Planning (resp., Weak Planning) can be solved, via the encoding $\overline{Strong}(P)$ (resp., $\overline{Weak}(P)$) in time $O(|\mathcal{S}|\cdot\|P\|)$, and by a properly implemented algorithm* STRONG PLAN *(resp.,* WEAK PLAN*), in time $O(|\mathcal{S}|\cdot|\Phi|)$.*

Simple implementations of the algorithms for strong and weak planning in (Cimatti *et al.* 2003) have time complexity $O(|S|\cdot|\Phi|)$, while more sophisticated ones have $O(\|P\|)$, i.e., linear time. For the special Horn encodings $\overline{Strong}^{+}(P)$ and $\overline{Weak}^{+}(P)$, we obtain the same time bound. They are closely related to the respective algorithms in (Cimatti *et al.* 2003) and may be viewed as declarative descriptions of the plan construction method. Nicely, an efficient implementation comes for free by the efficient algorithms for solving Horn theories.

As for the computational complexity of the planning problems, we note the following.[1]

Proposition 5 *Deciding whether a given planning problem $\langle\mathcal{D},\mathcal{I},\mathcal{G}\rangle$ has (i) a strong cyclic solution is* **P**-*complete (ii) a strong solution is* **P**-*complete, and (iii) a weak solution is* **NLOG**-*complete.*

The **P**-hardness results are an easy consequence of complexity results on k-maintainability (Baral & Eiter 2004). The **NLOG**-membership of weak solutions is explained by the fact that as shown above, this reduces to solving for each $s \in \mathcal{I}$ a Horn SAT instance (Theorem 3) that is also a 2-SAT instance, which is feasible in **NLOG**. The **NLOG**-hardness follows from a simple reduction from the canonical graph reachability problem. We finally note that exploiting Theorem 3, also computing some weak plan is feasible in nondeterministic logspace.

Extending the Approach to Other Goals

As we mentioned earlier, weak, plans, and strong cyclic plans can be expressed as $\mathsf{E}_\pi\Diamond p$, $\mathsf{A}_\pi\Diamond p$, and $\mathsf{A}_\pi\Box(\mathsf{E}_\pi\Diamond p)$ in language π-CTL*. Although we focus on these plans in this paper, the planning algorithm finding approach can be used to find algorithms for many other kinds of goals in π-CTL*. Such goals include:

- Maintenance Goals such as $\mathsf{A}_\pi\Box(\mathsf{E}_\pi\Box\Diamond p)$. Such goals are particularly relevant when possible exogenous actions can take the agent away from p, and it has to get back to p.
- Goals such as $\mathsf{A}\Box(\mathsf{E}_\pi\Diamond p)$, and $\mathsf{A}_\pi\Box(\mathsf{E}\Diamond p)$, where E and A correspond to exist path and all path regardless of whether they follow the policy under consideration or not.
- Goals of the kind: Reach p for sure; then reach q and so on. In this case the second 'reach' could mean either weak, strong or strong cyclic way of reaching. Here our approach can be used to generate multiple policies that are to be used depending on the status of achievement.
- Goals of the kind: Try to reach p (weak plan) and if at any point if p becomes unreachable then try to reach q.

[1] We could not find a reference for these results, which might be known to the specialists, though.

Conclusion

In this paper, we show that the methodology in (Baral & Eiter 2004) can be used to develop polynomial time planning algorithms for various kinds of problems in a non-deterministic domain, viz. for weak, strong, and strong cyclic planning. Small modifications to the algorithm obtained for strong cyclic planning, whose complexity is comparable to a sophisticated implementation of the Cimatti *et al.* algorithm, yield polynomial algorithms for strong and weak planning. Furthermore, simple, genuine Horn encodings give efficient (linear time) implementations of Cimatti *et al.*'s strong and weak plan construction method at an abstract level. We also show how strong cyclic planning can be declaratively done in non-monotonic logic programming, using an Answer Set Solver. By exploiting features of such solvers, a (most) preferred among multiple candidate plans, depending on criteria like deterministic actions, action preference, or action cost might be singled out.

It appears that the method described in this paper may be fruitfully applied to obtain polynomial-time planning algorithms for other kinds of goals in non-deterministic domains. Among them are several goals expressed in the language π-CTL* (Baral and Zhao 2004). Exploring this is part of our ongoing work.

Acknowledgements

This work was supported in part by grants NSF 0412000, NSF 0070463, FWF (Austrian Science Funds) project P-16536-N04 and EU project IST 2001-37004 WASP.

References

Baral, C., and Eiter, T. 2004. A polynomial time algorithm for constructing k-maintainable policies. In *Proc. ICAPS'04*, 720–730.

Baral, C., and Zhao, J. 2004 Goal specification in presence of non-deterministic actions. In *ECAI 2004*, pp. 272-277.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *AIJ* 147(1-2):35–84.

Dal Lago, U.; Pistore, M.; and Traverso, P. 2002. Planning with a language for extended goals. In *AAAI'02*, 447–454.

Dantsin, E.; Eiter, T.; Gottlob, G.; and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Comp. Surveys* 33(3):374–425.

Dowling, W., and Gallier, J. H. 1984. Linear-time algorithms for testing the satisfiability of propositional Horn theories. *Journal of Logic Programming* 3:267–284.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9:365–385.

Kautz, H., and Selman, B. 1992. Planning as Satisfiability. In *Proc. ECAI '92*, pp. 359–363.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2004. The DLV system for knowledge representation and reasoning. *ACM TOCL.* (To appear)

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and implementing the stable model semantics. *Art. Intelligence* 138:181–234.