Lecture 6: AI Planning

Outline

1. Planning problems
2. Restricted plans and ASP
3. Improvements on the encoding

1. PLANNING PROBLEMS

- *Planning problems* from the area of *artificial intelligence* (AI) form a computationally challenging application domain for ASP.
- It is difficult to tailor special-purpose algorithms designed to solve planning problems for new application domains.
- A more flexible representation can be obtained by describing planning problems in terms of (propositional) logic.
- Logical descriptions tend to become large due to a *frame problem* or the law of *inertia*: things do not change without a cause.
- Logic programs under stable models provide a promising way to tackle the complexity of expressing knowledge of this kind.

Formal Definition

A planning problem is a quadruple $\langle \mathcal{D}, \mathcal{O}, S_0, S_1 \rangle$ defined as follows:

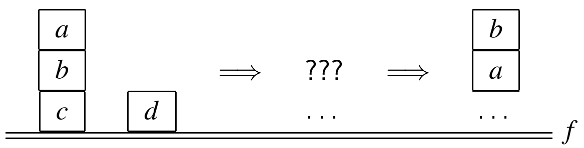
- The members of each *finite domain* $D \in \mathcal{D}$ have been uniquely named with a set of *naming constants* $\{a, b, c, \ldots\}$.
- Every *operator* $O(x, y, z, \ldots) \in \mathcal{O}$ consists of
  1. domain definitions for its variables $x : D_1$, $y : D_2$, $z : D_3$, $\ldots$, and
  2. sets of *preconditions* $\mathrm{Pre}(O)$ and *postconditions* $\mathrm{Post}(O)$ which are sets of atomic, well-formed, and typed formulas built from relation symbols P, Q, $\ldots$, variables x, y, z, $\ldots$ and constants $a, b, c, \ldots$ associated with specific domains.
- The *initial situation* S_0 and *final situation* S_1 are sets of ground (variable-free) atomic formulas, i.e., *atomic sentences*.

Example: Blocks' World (I)

- The domain Block is named by constants $\{a, b, c, d\}$.
- There is an additional object, *the floor* denoted by f, which cannot be moved and which can hold multiple blocks.
- The initial situation is given by

  $$S_0 = \{\mathsf{On}(a,b), \mathsf{On}(b,c), \mathsf{On}(c,f), \mathsf{On}(d,f), \mathsf{Clear}(a), \mathsf{Clear}(d)\}.$$

- The goal is determined by the set of conditions $S_1 = \{\mathsf{On}(b,a)\}$.



Example: Blocks' World (II)

- Block is the domain of variables x, y, and z below.
- There is only one operator MOVE(x,y,z):
  1. Pre(MOVE$(x,y,z)) = \{$Clear(x), On(x,y), Clear$(z)\}$ and
  2. Post(MOVE$(x,y,z)) = \{$On(x,z), Clear(x), Clear$(y)\}$.
- For moving blocks on the floor, we need MOVE(x,y,f) with $y \neq f$:
  1. Pre(MOVE$(x,y,f)) = \{$Clear(x), On$(x,y)\}$ and
  2. Post(MOVE$(x,y,f)) = \{$On(x,f), Clear(x), Clear$(y)\}$.
- Another special case is MOVE(x,f,z) where $z \neq f$:
  1. Pre(MOVE$(x,f,z)) = \{$Clear(x), On(x,f), Clear$(z)\}$ and
  2. Post(MOVE$(x,f,z)) = \{$On(x,z), Clear$(x)\}$.

Plans as Solutions

- An operator $O(x,y,z,\ldots) \in \mathcal{O}$ can be instantiated by applying a *substitution* $\sigma = \{x/c_1, y/c_2, z/c_3, \ldots\}$ where $c_1, c_2, c_3, \ldots$ are some naming constants from the respective domains $D_1, D_2, D_3, \ldots$.
- An individual *action* $O\sigma$, i.e., a ground instance of an operator $O(\vec{x}) \in \mathcal{O}$, can take place in a situation S where $\mathrm{Pre}(O)\sigma \subseteq S$.
- The outcome of performing an action in S is a revised situation

$$S' = (S \setminus \mathrm{Pre}(O)\sigma) \cup \mathrm{Post}(O)\sigma$$

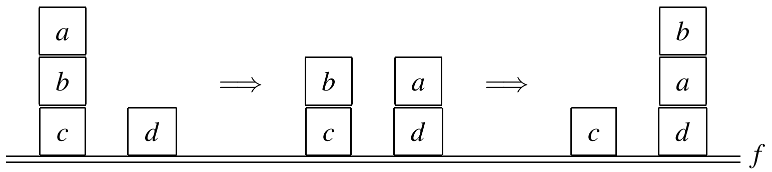
to be denoted by $S \overset{O\sigma}{\Longrightarrow} S'$ in the sequel.

Definition. A solution to a planning problem $\langle \mathcal{D}, \mathcal{O}, S_0, S_1 \rangle$ is a sequence of actions $O_1\sigma_1, \ldots, O_n\sigma_n$, i.e., a *plan* that turns the initial situation S_0 into a situation S satisfying $S_1 \subseteq S$: $S_0 \overset{O_1\sigma_1}{\Longrightarrow} \cdots \overset{O_n\sigma_n}{\Longrightarrow} S$.

Example: Blocks' World (III)

The sequence of actions MOVE(a,b,d) and MOVE(b,c,a) forms a valid plan and a solution to the problem in question:

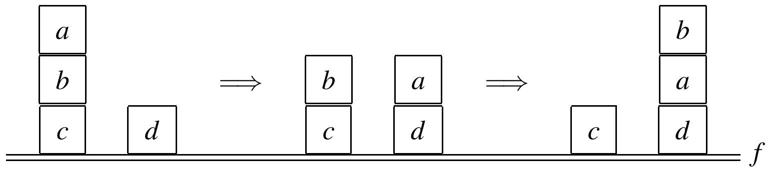
$$\begin{array}{rl} & \{\mathrm{On}(a,b), \mathrm{On}(b,c), \mathrm{On}(c,f), \mathrm{On}(d,f), \mathrm{Clear}(a), \mathrm{Clear}(d)\} \\ \overset{\mathrm{MOVE}(a,b,d)}{\Longrightarrow} & \{\mathrm{On}(b,c), \mathrm{On}(c,f), \mathrm{On}(a,d), \mathrm{On}(d,f), \mathrm{Clear}(a), \mathrm{Clear}(b)\} \\ \overset{\mathrm{MOVE}(b,c,a)}{\Longrightarrow} & \{\mathrm{On}(c,f), \mathrm{On}(b,a), \mathrm{On}(a,d), \mathrm{On}(d,f), \mathrm{Clear}(c), \mathrm{Clear}(b)\}. \end{array}$$



Example: Blocks' World (IV)

The pre- and postconditions for the actions involved are:

1. Pre(MOVE$(a,b,d)) = \{$Clear(a), On(a,b), Clear$(d)\}$
   Post(MOVE$(a,b,d)) = \{$On(a,d), Clear(a), Clear$(b)\}$
2. Pre(MOVE$(b,c,a)) = \{$Clear(b), On(b,c), Clear$(a)\}$
   Post(MOVE$(b,c,a)) = \{$On(b,a), Clear(b), Clear$(c)\}$



Remarks on Computational Complexity

- It is computationally very demanding to solve planning problems.
- Consider the following *decision problems*:
  1. PLANSAT: does the given planning problem $\langle \mathcal{D}, O, S_0, S_1 \rangle$ have a solution?
  2. PLANMIN: does the given planning problem $\langle \mathcal{D}, O, S_0, S_1 \rangle$ have a solution of length k—the limit k being part of the input?
- PLANSAT and PLANMIN are PSPACE-complete.

Remark. A way to govern computational complexity is to limit plan length to polynomial with respect to the length of the instance—the limit k is given in base 1. Such restricted problems are NP-complete.

2. RESTRICTED PLANS AND ASP

- Consider the following NP-complete decision problem STABLE:

  Does the given normal logic program P have a stable model?
- As a consequence, any instance $\langle \mathcal{D}, O, S_1, S_2, 1_k \rangle$ of the *restricted* planning problem can be *reduced* to an instance of STABLE.
- The complexity theory behind NP-completeness is only concerned with the preservation of **yes**/**no**-answers under reductions.
- A tight correspondence of answer sets and plans can be achieved when restricted planning problems are represented in ASP.
- In fact, the minimality of stable models and their strong groundedness simplify the representation of planning problems.

Overall Goals for the Translation

- The translation of a restricted planning problem $\langle \mathcal{D}, O, S_0, S_1, 1_k \rangle$ is a normal logic program $\mathrm{LPlan}_k(\mathcal{D}, O, S_0, S_1)$ such that

  the problem $\langle \mathcal{D}, O, S_0, S_1 \rangle$ has a solution of length at most k

  $\iff$ $\mathrm{LPlan}_k(\mathcal{D}, O, S_0, S_1)$ has a stable model.
- Such a representation $\mathrm{LPlan}_k(\mathcal{D}, O, S_0, S_1)$ is called *constructive* if there is a polytime algorithm for extracting a solution, i.e., a plan for $\langle \mathcal{D}, O, S_0, S_1, 1_k \rangle$ from a model $M \in \mathrm{SM}(\mathrm{LPlan}_k(\mathcal{D}, O, S_0, S_1))$.
- We aim at a straightforward representation using which plans can be recovered as simple projections of answer sets.

Linear Plans

- In the sequel, our approach is to describe the solutions

  $$O_1\sigma_1, \ldots, O_n\sigma_n$$

  of a restricted planning problem $\langle \mathcal{D}, O, S_0, S_1, 1_k \rangle$ with rules.
- First, a *linear* notion of time will be used: exactly one action $O_t\sigma_t$ will be accomplished at each point of time $t \in \{0, 1, \ldots, k-1\}$.
- An extra variable for time, namely t, is added to every relation symbol and operator: $\mathrm{Clear}(x,t)$, $\mathrm{On}(x,y,t)$, and $\mathrm{MOVE}(x,y,z,t)$.
- For relation symbols, the time t varies in the range $0, \ldots, k$ whereas for operators it is in the range $0, \ldots, k-1$.

Describing Restricted Plans

The description of restricted plans comes in five parts:

1. Determining the initial situation ($t = 0$): $\{P(\vec{c}, 0) \mid P(\vec{c}) \in S_0\}$.
2. Things that must hold in the end ($t = k$): $\{P(\vec{c}, k) \mid P(\vec{c}) \in S_1\}$.
3. An action $O\sigma$ is performed at time t if its preconditions are satisfied and there are *no exceptions* to it. Consequently, things that become true $(\mathrm{Post}(O)\sigma \setminus \mathrm{Pre}(O)\sigma)$ hold at time $t+1$.
4. *Frame axioms*: if $P(\vec{c})$ holds at time t and no action falsifies it at time t, it will hold at time $t+1$ as well.
5. *At most one* action $O_t\sigma_t$ is performed at each time step t, i.e., $O_t\sigma_t$ causes an *exception* to all other actions at time t.

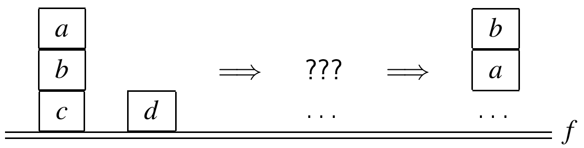
An Encoding of Blocks' World (I)

1. Initial situation S_0:

   On$(a,b,0)$. On$(b,c,0)$. On$(c,f,0)$. On$(d,f,0)$.
   Clear$(a,0)$. Clear$(d,0)$.

2. Final situation S_1 with the parameter k:

   $\leftarrow \sim$On(b,a,k).



An Encoding of Blocks' World (II)

- The domain Block and its extension Object with f:

  Block(a). Block(b). Block(c). Block(d).
  Object$(x) \leftarrow$ Block(x). Object(f).

- A domain for triples of objects potentially subject to moves:

  Diff$(x,y,z) \leftarrow \sim(x=y), \sim(x=z), \sim(y=z)$,
  Block(x), Object$(y;z)$.

- Specify time points and objects that can hold a block:

  Time(0). ... Time(k).
  CanHold$(z,t) \leftarrow$ Clear(z,t), Block(z), Time(t).
  CanHold$(f,t) \leftarrow$ Time(t).

An Encoding of Blocks' World (III)

3. Applications of the operator MOVE:

   MOVE$(x,y,z,t) \leftarrow$ Clear(x,t), On(x,y,t), CanHold(z,t),
   $\sim$DeniedMOVE(x,y,z,t), Diff(x,y,z), Time(t), $t < k$.
   On$(x,z,t+1) \leftarrow$ MOVE(x,y,z,t),
   Diff(x,y,z), Time$(t;t+1)$.
   Clear$(y,t+1) \leftarrow$ MOVE(x,y,z,t),
   Diff(x,y,z), $\sim(y=f)$, Time$(t;t+1)$.

Remark. No rule for Clear$(x,t+1)$ is needed as Clear(x,t) is a precondition of MOVE(x,y,z,t) and the respective frame axiom will imply Clear$(x,t+1)$.

An Encoding of Blocks' World (IV)

4. Frame axioms cover atomic sentences that remain true:

$$\mathsf{On}(x,y,t+1) \leftarrow \mathsf{On}(x,y,t), \sim\mathsf{RemoveOn}(x,y,t),$$
$$\mathsf{Block}(x), \mathsf{Object}(y), \mathsf{Time}(t;t+1).$$
$$\mathsf{RemoveOn}(x,y,t) \leftarrow \mathsf{MOVE}(x,y,z,t),$$
$$\mathsf{Diff}(x,y,z), \mathsf{Time}(t),\ \ t<k.$$

$$\mathsf{Clear}(x,t+1) \leftarrow \mathsf{Clear}(x,t), \sim\mathsf{RemoveClear}(x,t),$$
$$\mathsf{Block}(x), \mathsf{Time}(t;t+1).$$
$$\mathsf{RemoveClear}(z,t) \leftarrow \mathsf{MOVE}(x,y,z,t),$$
$$\mathsf{Diff}(x,y,z), \sim(z=f), \mathsf{Time}(t),\ \ t<k.$$

An Encoding of Blocks' World (V)

5. Enforcing the *linearity* of plans, i.e., only one action can be performed at a time which is expressed using exceptions.

$$\mathsf{DeniedMOVE}(x,y,z,t) \leftarrow \mathsf{MOVE}(u,v,w,t), \sim(x=u),$$
$$\mathsf{Diff}(x,y,z), \mathsf{Diff}(u,v,w), \mathsf{Time}(t),\ \ t<k.$$
$$\mathsf{DeniedMOVE}(x,y,z,t) \leftarrow \mathsf{MOVE}(u,v,w,t), \sim(y=v),$$
$$\mathsf{Diff}(x,y,z), \mathsf{Diff}(u,v,w), \mathsf{Time}(t),\ \ t<k.$$
$$\mathsf{DeniedMOVE}(x,y,z,t) \leftarrow \mathsf{MOVE}(u,v,w,t), \sim(z=w),$$
$$\mathsf{Diff}(x,y,z), \mathsf{Diff}(u,v,w), \mathsf{Time}(t),\ \ t<k.$$

Remark. The number of rules that encode exceptions to moves grows fast as the number of blocks grows (to be addressed below).

An Encoding of Blocks' World (VI)

Preceding rules state that exactly one move is made at each time step.

5. By allowing *self-exceptions*, which effectuate a form of *choice*, we capture the *at most one action* aspect of the specification:

$$\mathsf{DeniedMOVE}(x,y,z,t) \leftarrow \sim\mathsf{MOVE}(x,y,z,t),$$
$$\mathsf{Diff}(x,y,z), \mathsf{Time}(t),\ \ t<k.$$

Alternatively, any of the final situations can cause an exception:

$$\mathsf{DeniedMOVE}(x,y,z,t) \leftarrow \mathsf{GoalReached}(t),$$
$$\mathsf{Diff}(x,y,z), \mathsf{Time}(t),\ \ t<k.$$
$$\mathsf{GoalReached}(t) \leftarrow \mathsf{On}(b,a,t), \mathsf{Time}(t),\ \ t<k.$$

Remark. If there were additional operators in this domain, also inter-operator conflicts would have to be formalized using rules of this kind.

3. IMPROVEMENTS ON THE ENCODING

- The length of the resulting ground program can be decreased by splitting operators, viewed as special relations, in parts.
- The same technique can be applied to other relation symbols.
- Yet another strategy is to give up the linearity of plans: mutually independent actions can be performed concurrently.
- Plans can be further enhanced in terms of additional constraints.

Example. Forbid two subsequent moves of the same block:

$$\leftarrow \mathsf{MOVE}(x,y,z,t), \mathsf{MOVE}(x,z,u,t+1),$$
$$\mathsf{Diff}(x,y,z), \mathsf{Diff}(x,z,u), \mathsf{Time}(t;t+1).$$

Splitting Operators

- Operators with multiple arguments increase the size of the resulting encoding and, in particular, its ground instance.
- A way to govern combinatorial explosion in the encoding is to split the respective relations in component relations as far as possible.

Example. In the Blocks' world domain, the relation MOVE(x,y,z,t) can be split into TGT(x,t), SRC(y,t), and DST(z,t) using t as a key:

$$\mathrm{MOVE}(x,y,z,t) \leftarrow \mathrm{TGT}(x,t), \mathrm{SRC}(y,t), \mathrm{DST}(z,t),$$
$$\mathrm{Diff}(x,y;z), \mathrm{Time}(t),\ t < k.$$

But this rule is not included in the program: MOVE(x,y,z,t) is defined *implicitly* in terms of TGT(x,t), SRC(y,t), and DST(z,t).

Definition of MOVE after Splitting (I)

1. Selection of the action to be performed at time t:

$$\mathrm{Diff}(x,y) \leftarrow \mathrm{Block}(x), \mathrm{Object}(y), \sim(x=y).$$
$$\mathrm{TGT}(x,t) \leftarrow \mathrm{Clear}(x,t), \mathrm{On}(x,y,t), \sim\mathrm{DeniedTGT}(x,t),$$
$$\mathrm{Diff}(x,y), \mathrm{Time}(t),\ t < k.$$
$$\mathrm{SomeTGT}(t) \leftarrow \mathrm{TGT}(x,t), \mathrm{Block}(x), \mathrm{Time}(t),\ t < k.$$
$$\mathrm{SRC}(y,t) \leftarrow \mathrm{TGT}(x,t), \mathrm{On}(x,y,t), \mathrm{Diff}(x,y), \mathrm{Time}(t),\ t < k.$$
$$\mathrm{DST}(z,t) \leftarrow \mathrm{CanHold}(z,t), \sim\mathrm{DeniedDST}(z,t),$$
$$\mathrm{Object}(z), \mathrm{Time}(t),\ t < k.$$

2. Things that become true once this action is performed:

$$\mathrm{On}(x,z,t+1) \leftarrow \mathrm{TGT}(x,t), \mathrm{DST}(z,t), \mathrm{Diff}(x,z), \mathrm{Time}(t;t+1).$$
$$\mathrm{Clear}(y,t+1) \leftarrow \mathrm{SRC}(y,t), \mathrm{Block}(y), \sim(y=f), \mathrm{Time}(t;t+1).$$

Definition of MOVE after Splitting (II)

2. Avoiding conflicts with other actions (uniqueness of moves):

$$\mathrm{DeniedTGT}(x,t) \leftarrow \mathrm{TGT}(y,t),$$
$$\mathrm{Block}(x;y), \sim(y=x), \mathrm{Time}(t),\ t < k.$$
$$\mathrm{DeniedTGT}(x,t) \leftarrow \sim\mathrm{TGT}(x,t), \mathrm{Block}(x), \mathrm{Time}(t),\ t < k.$$
$$\mathrm{DeniedDST}(z,t) \leftarrow \mathrm{DST}(y,t),$$
$$\sim(z=y), \mathrm{Object}(y;z), \mathrm{Time}(t),\ t < k.$$
$$\mathrm{DeniedDST}(x,t) \leftarrow \mathrm{TGT}(x,t), \mathrm{Block}(x), \mathrm{Time}(t),\ t < k.$$
$$\mathrm{DeniedDST}(y,t) \leftarrow \mathrm{SRC}(y,t), \mathrm{Object}(y), \mathrm{Time}(t),\ t < k.$$
$$\mathrm{DeniedDST}(z,t) \leftarrow \sim\mathrm{SomeTGT}(t), \mathrm{Time}(t),\ t < k.$$

Remark. The predicate SomeTGT(t) is used to prohibit the choice of the destination object whenever no block is going to be moved.

Positive Effects of Splitting

k	n	r_1	t_1	r_2	t_2
1	0	5764	0.092	199	0.004
2	2	11458	0.180	356	0.004
3	16	17152	0.288	513	0.012
4	107	22846	0.528	670	0.048
5	678	28540	1.520	827	0.192
6	4249	34234	5.560	984	1.024

n: Number of plans

r_i: Nuber of rules in the ground program (1=before, 2=after)

t_i: Running time of `smodels` for computing all plans

Partial vs. Linear Orders of Time

- By allowing several concurrent and mutually independent actions simultaneously, we obtain plans that are partial orders of actions.
- Savings are expected as the required number of time steps is likely to be smaller than the length of the respective linear plan.
- Given a partial plan, a linear plan is obtained by taking any linear order of actions which is compatible with the partial order.

$$\begin{array}{ccccc} & \nearrow\!\le & \mathsf{MOVE}(b,f,a) & \le\!\searrow & \\ \mathsf{MOVE}(a,b,c) & & \le & & \mathsf{MOVE}(b,a,d) \quad \Longrightarrow \\ & \searrow\!\le & \mathsf{MOVE}(d,f,e) & \le\!\nearrow & \end{array}$$

$$\begin{cases} \mathsf{MOVE}(a,b,c) < \mathsf{MOVE}(b,f,a) < \mathsf{MOVE}(d,f,e) < \mathsf{MOVE}(b,a,d) \\ \mathsf{MOVE}(a,b,c) < \mathsf{MOVE}(d,f,e) < \mathsf{MOVE}(b,f,a) < \mathsf{MOVE}(b,a,d) \end{cases}$$

Blocks' World Strikes Again

- The concurrent execution of moves is forbidden only in case of a real conflict (two actions share a resource).
- A block cannot be moved to two different destinations:

$$\mathsf{DeniedMOVE}(x,y,z,t) \leftarrow \mathsf{MOVE}(x,y,u,t), \sim(z=u), \mathsf{Diff}(x,y,z), \mathsf{Diff}(x,y,u), \mathsf{Time}(t),\ t<k.$$

- Two different blocks cannot be moved to the same destination:

$$\mathsf{DeniedMOVE}(x,y,z,t) \leftarrow \mathsf{MOVE}(u,v,z,t), \sim(x=u), \mathsf{Diff}(x,y,z), \mathsf{Diff}(u,v,z), \mathsf{Time}(t),\ t<k.$$

- A block cannot be moved to a destination that is being moved:

$$\mathsf{DeniedMOVE}(x,y,z,t) \leftarrow \mathsf{MOVE}(z,u,v,t), \mathsf{Difff}(x,y,z), \mathsf{Diff}(z,u,v), \mathsf{Time}(t),\ t<k.$$

OBJECTIVES

- You understand the definition of the planning problem as well as that of its solutions.
- You are aware of the high computational complexity involved in planning problems in general.
- You are able to solve a simple planning problem
  1. by representing it as a normal logic program,
  2. by computing answer sets for the program, and
  3. by extracting concrete plans from the answer sets found.

TIME TO PONDER

Partial plans are not directly applicable if the operators involved are split in the way described above.

- Why is this the case?
- What kind of modifications are necessary in order to apply the two techniques simultaneously?