

Planning III: planning using propositional logic

We've seen that plans might be extracted from a knowledge base via *theorem proving*, using *first order logic (FOL)* and *situation calculus*.

BUT: this might be computationally infeasible for realistic problems.

Sophisticated techniques are available for testing *satisfiability* in *propositional logic*, and these have also been applied to planning.

The basic idea is to attempt to find a model of a sentence having the form

$$\begin{aligned} & \text{description of start state} \\ & \wedge \text{descriptions of the possible actions} \\ & \wedge \text{description of goal} \end{aligned}$$

We attempt to construct this sentence such that:

- If M is a model of the sentence then M assigns **true** to a proposition if and only if it is in the plan.
- Any assignment denoting an incorrect plan will not be a model as the goal description will not be **true**.
- The sentence is unsatisfiable if no plan exists.

1

Propositional logic for planning

Two roof-climbers want to *swap places*:

Start state:

$$\begin{aligned} S = & At^0(a, \text{spire}) \wedge At^0(b, \text{ground}) \\ & \wedge \neg At^0(a, \text{ground}) \wedge \neg At^0(b, \text{spire}) \end{aligned}$$



Remember that an expression such as $At^0(a, \text{spire})$ is a *proposition*. The super-scripted number now denotes time.

2

Propositional logic for planning

Goal:

$$\begin{aligned} G = & At^i(a, \text{ground}) \wedge At^i(b, \text{spire}) \\ & \wedge \neg At^i(a, \text{spire}) \wedge \neg At^i(b, \text{ground}) \end{aligned}$$

Actions: can be introduced using the equivalent of successor-state axioms

$$\begin{aligned} At^1(a, \text{ground}) \leftrightarrow \\ & (At^0(a, \text{ground}) \wedge \neg Move^0(a, \text{ground}, \text{spire})) \\ & \vee (At^0(a, \text{spire}) \wedge Move^0(a, \text{spire}, \text{ground})) \end{aligned} \quad (1)$$

Denote by A the collection of all such axioms.

3

Propositional logic for planning

We will now find that $S \wedge A \wedge G$ has a model in which $Move^0(a, \text{spire}, \text{ground})$ and $Move^0(b, \text{ground}, \text{spire})$ are **true** while all remaining actions are **false**.

In more realistic planning problems we will clearly not know in advance at what time the goal might expect to be achieved.

We therefore:

- Loop through possible final times T .
- Generate a goal for time T and actions up to time T .
- Try to find a model and extract a plan.
- Until a plan is obtained or we hit some maximum time.

4

Propositional logic for planning

Unfortunately there is a problem—we may, if considerable care is not applied, also be able to obtain less sensible plans.

In the current example

$$\text{Move}^0(\text{b, ground, spire}) = \text{true}$$

$$\text{Move}^0(\text{a, spire, ground}) = \text{true}$$

$$\text{Move}^0(\text{a, ground, spire}) = \text{true}$$

is a model, because the successor-state axiom (1) does not in fact preclude the application of $\text{Move}^0(\text{a, ground, spire})$.

We need a *precondition axiom*

$$\text{Move}^i(\text{a, ground, spire}) \rightarrow \text{At}^i(\text{a, ground})$$

and so on.

5

Propositional logic for planning

Life becomes more complicated still if a third location is added: *hospital*.

$$\text{Move}^0(\text{a, spire, ground}) \wedge \text{Move}^0(\text{a, spire, hospital})$$

is perfectly valid and so we need to specify that he can't move to two places simultaneously

$$\neg(\text{Move}^i(\text{a, spire, ground}) \wedge \text{Move}^i(\text{a, spire, hospital}))$$

$$\neg(\text{Move}^i(\text{a, ground, spire}) \wedge \text{Move}^i(\text{a, ground, hospital}))$$

⋮

and so on.

These are *action-exclusion* axioms.

Unfortunately they will tend to produce *totally-ordered* rather than *partially-ordered* plans.

6

Propositional logic for planning

Alternatively:

1. Prevent actions occurring together if one negates the effect or precondition of the other.
2. Or, specify that something can't be in two places simultaneously

$$\neg(\text{At}^i(x, 11) \wedge \text{At}^i(x, 12))$$

for all combinations of x, i and $11 \neq 12$.

This is an example of a *state constraint*.

Clearly this process can become very complex, but there are techniques to help deal with this.

7

Review of constraint satisfaction problems (CSPs)

Recall that in a CSP we have:

- A set of n variables V_1, V_2, \dots, V_n .
- For each V_i a *domain* D_i specifying the values that V_i can take.
- A set of m constraints C_1, C_2, \dots, C_m .

Each constraint C_i involves a set of variables and specifies an *allowable collection of values*.

- A *state* is an assignment of specific values to some or all of the variables.
- An assignment is *consistent* if it violates no constraints.
- An assignment is *complete* if it gives a value to every variable.

A *solution* is a consistent and complete assignment.

8

The state-variable representation

Another planning language: the *state-variable representation*.

Things of interest such as people, places, objects *etc* are divided into *domains*:

$$\mathcal{D}_1 = \{\text{climber1, climber2}\}$$

$$\mathcal{D}_2 = \{\text{home, jokeShop, hardwareStore, pavement, spire, hospital}\}$$

$$\mathcal{D}_3 = \{\text{rope, gorilla}\}$$

Part of the specification of a planning problem involves stating which domain a particular item is in. For example

$$\mathcal{D}_1(\text{climber1})$$

and so on.

Relations and functions have arguments chosen from unions of these domains.

$$\text{above} \subseteq \mathcal{D}_1^{\text{above}} \times \mathcal{D}_2^{\text{above}}$$

is a relation. The $\mathcal{D}_i^{\text{above}}$ are unions of one or more \mathcal{D}_i .

Note: \mathcal{D} is used for domains in the state-variable representation. D is used for domains in CSPs.

The state-variable representation

The relation *above* is in fact a *rigid relation (RR)*, as it is unchanging: it does not depend upon *state*. (Remember *fluents* in situation calculus?)

Similarly, we have *functions*

$$\text{at}(x_1, s) : \mathcal{D}_1^{\text{at}} \times S \rightarrow \mathcal{D}^{\text{at}}.$$

Here, $\text{at}(x, s)$ is a *state-variable*. The domain $\mathcal{D}_1^{\text{at}}$ and range \mathcal{D}^{at} are unions of one or more \mathcal{D}_i . In general these can have multiple parameters

$$\text{sv}(x_1, \dots, x_n, s) : \mathcal{D}_1^{\text{sv}} \times \dots \times \mathcal{D}_n^{\text{sv}} \times S \rightarrow \mathcal{D}^{\text{sv}}.$$

A state-variable denotes assertions such as

$$\text{at}(\text{gorilla}, s) = \text{jokeShop}$$

where s denotes a *state* and the set S of all states will be defined later.

The state variable allows things such as locations to change—again, much like *fluents* in the situation calculus.

Variables appearing in relations and functions are considered to be *typed*.

The state-variable representation

Note:

- For properties such as a *location* a function might be considerably more suitable than a relation.
- For locations, everything has to be *somewhere* and it can only be in *one place at a time*.

So a function is perfect and immediately solves some of the problems seen earlier.

The state-variable representation

Actions as usual, have a *name*, a *set of preconditions* and a *set of effects*.

- *Names* are unique, and followed by a list of variables involved in the action.
- *Preconditions* are expressions involving state variables and relations.
- *Effects* are assignments to state variables.

For example:

$\text{buy}(x, y, l)$	
Preconditions	$\text{at}(x, s) = l$ $\text{sells}(l, y)$ $\text{has}(y, s) = l$
Effects	$\text{has}(y, s) = x$

The state-variable representation

Goals are sets of *expressions* involving *state variables*.

For example:

Goal:
at(climber, s) = home
has(ropes, s) = climber
at(gorilla, s) = spire

From now on we will generally suppress the state s when writing state variables.

13

The state-variable representation

A *state* as just a statement of what values the state variables take at a given time.

$$s = \{ \begin{array}{l} \text{has(gorilla)} = \text{jokeShop} \\ \text{has(firstAidKit)} = \text{climber2} \\ \text{has(ropes)} = \text{climber2} \\ \vdots \\ \text{at(climber1)} = \text{jokeShop} \\ \text{at(climber2)} = \text{spire} \\ \vdots \end{array} \}$$

- For each state variable sv consider all ground instances, such as $sv(\text{climber}, \text{rope})$, with arguments *consistent* with the *rigid relations*. Define X to be the set of all such ground instances.

- A state s is then just a set

$$s = \{(v = c) | v \in X\}$$

where c is in the range of v .

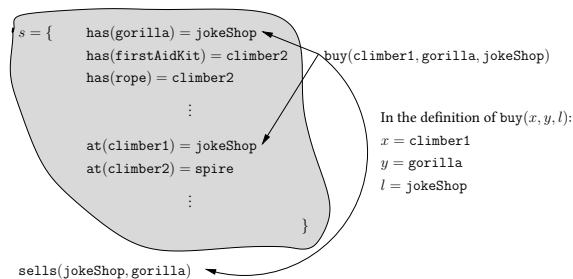
This allows us to define the *effect of an action*.

A planning problem also needs a *start state* s_0 , which can be defined in this way.

14

The state-variable representation

Considering all the *ground actions consistent with the rigid relations*:



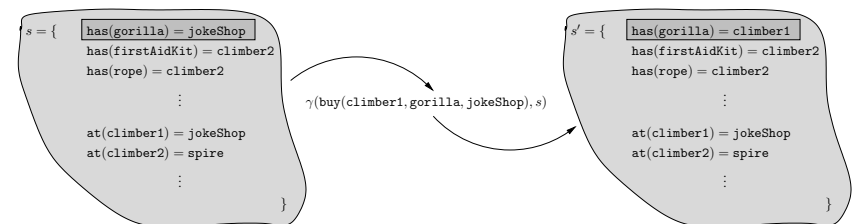
- An action is *applicable in s* if all expressions $v = c$ appearing in the set of preconditions also appear in s .
- As there is no rigid relation $sells(\text{jokeShop}, \text{fruitBats})$ we would *not* consider an action such as $buy(\text{climber1}, \text{fruitBats}, \text{jokeShop})$ —it is not *consistent with the rigid relations*.

15

The state-variable representation

Finally, there is a function γ that maps a state and an action to a new state

$$\gamma(s, a) = s'$$



Specifically, we have

$$\gamma(s, a) = \{(v = c) | v \in X\}$$

where either c is specified in an effect of a , or otherwise $v = c$ is a member of s .

Note: the definition of γ implicitly solves the *frame problem*.

16

The state-variable representation

A *solution* to a planning problem is a sequence (a_0, a_1, \dots, a_n) of actions such that...

- a_0 is applicable in s_0 and for each i , a_i is applicable in $s_i = \gamma(s_{i-1}, a_{i-1})$.
- For each goal g we have

$$g \in \gamma(s_n, a_n).$$

What we need now is a method for *transforming* a problem described in this language into a CSP.

We'll once again do this for a fixed upper limit T on the number of steps in the plan.

17

Converting to a CSP

Step 1: encode *actions* as *CSP variables*.

For each time step t where $0 \leq t \leq T - 1$, the CSP has a variable

$$\text{action}^t$$

with domain

$$D^{\text{action}^t} = \{a \mid a \text{ is the ground instance of an action}\} \cup \{\text{none}\}$$

Example: at some point in searching for a plan we might attempt to find the solution to the corresponding CSP involving

$$\text{action}^5 = \text{attach}(\text{gorilla}, \text{spire})$$

WARNING: be careful in what follows to distinguish between *state variables*, *actions etc* in the planning problem and *variables* in the CSP.

18

Converting to a CSP

Step 2: encode *ground state variables* as *CSP variables*, with a complete copy of all the state variables *for each time step*.

So, for each t where $0 \leq t \leq T$ we have a CSP variable

$$\text{sv}_i^t(c_1, \dots, c_n)$$

with domain $D = \mathcal{D}^{\text{sv}_i}$. (That is, the *domain* of the CSP variable is the *range* of the state variable.)

Example: at some point in searching for a plan we might attempt to find the solution to the corresponding CSP involving

$$\text{location}^9(\text{climber1}) = \text{hospital}.$$

19

Converting to a CSP

Step 3: encode the *preconditions for actions in the planning problem* as *constraints in the CSP problem*.

For each time step t and for each ground action $a(c_1, \dots, c_n)$ with arguments *consistent with the rigid relations in its preconditions*:

For a precondition of the form $\text{sv}_i = v$ include constraint pairs

$$\begin{aligned} (\text{action}^t = a(c_1, \dots, c_n), \\ \text{sv}_i^t = v) \end{aligned}$$

Example: consider the action $\text{buy}(x, y, l)$ introduced above, and having the preconditions $\text{at}(x) = l$, $\text{sells}(l, y)$ and $\text{has}(y) = l$.

Assume $\text{sells}(y, l)$ is only true for

$$l = \text{jokeShop}$$

and

$$y = \text{gorilla}$$

so we only consider these values for l and y . Then for each time step t we have the constraints...

20

Converting to a CSP

$action^t = \text{buy}(\text{climber1}, \text{gorilla}, \text{jokeShop})$ paired with $at^t(\text{climber1}) = \text{jokeShop}$
$action^t = \text{buy}(\text{climber1}, \text{gorilla}, \text{jokeShop})$ paired with $has^t(\text{gorilla}) = \text{jokeShop}$
$action^t = \text{buy}(\text{climber2}, \text{gorilla}, \text{jokeShop})$ paired with $at^t(\text{climber2}) = \text{jokeShop}$
$action^t = \text{buy}(\text{climber2}, \text{gorilla}, \text{jokeShop})$ paired with $has^t(\text{gorilla}) = \text{jokeShop}$
and so on...

21

Converting to a CSP

Step 4: encode the effects of actions in the planning problem as constraints in the CSP problem.

For each time step t and for each ground action $a(c_1, \dots, c_n)$ with arguments consistent with the rigid relations in its preconditions:

For an effect of the form $sv_i = v$ include constraint pairs

$$(action^t = a(c_1, \dots, c_n), \\ sv_i^{t+1} = v)$$

Example: continuing with the previous example, we will include constraints

$action^t = \text{buy}(\text{climber1}, \text{gorilla}, \text{jokeShop})$ paired with $has^{t+1}(\text{gorilla}) = \text{climber1}$
$action^t = \text{buy}(\text{climber2}, \text{gorilla}, \text{jokeShop})$ paired with $has^{t+1}(\text{gorilla}) = \text{climber2}$
and so on...

22

Converting to a CSP

Step 5: encode the frame axioms as constraints in the CSP problem.

An action must not change things not appearing in its effects. So:

For:

1. Each time step t .
2. Each ground action $a(c_1, \dots, c_n)$ with arguments consistent with the rigid relations in its preconditions.
3. Each sv_i that does not appear in the effects of a , and each $v \in \mathcal{D}^{sv_i}$

include in the CSP the ternary constraint

$$(action^t = a(c_1, \dots, c_n), \\ sv_i^t = v, \\ sv_i^{t+1} = v).$$

23

Finding a plan

Finally, having encoded a planning problem into a CSP, we solve the CSP.

The scheme has the following property:

A solution to the planning problem with at most T steps exists if and only if there is a solution to the corresponding CSP.

Assume the CSP has a solution.

Then we can extract a plan simply by looking at the values assigned to the $action^t$ variables in the solution of the CSP.

It is also the case that:

There is a solution to the planning problem with at most T steps if and only if there is a solution to the corresponding CSP from which the solution can be extracted in this way.

For a proof see:

Automated Planning: Theory and Practice

Malik Ghallab, Dana Nau and Paolo Traverso. Morgan Kaufmann 2004.

24