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

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

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.

Propositional logic for planning

Two roof-climbers want to *swap places*:

Start state:

$$S = \texttt{At}^0(\texttt{a}, \texttt{spire}) \land \texttt{At}^0(\texttt{b}, \texttt{ground}) \\ \land \neg\texttt{At}^0(\texttt{a}, \texttt{ground}) \land \neg\texttt{At}^0(\texttt{b}, \texttt{spire})$$

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

Propositional logic for planning

Goal:

$$G = \texttt{At}^i(\texttt{a}, \texttt{ground}) \wedge \texttt{At}^i(\texttt{b}, \texttt{spire}) \wedge \neg\texttt{At}^i(\texttt{a}, \texttt{spire}) \wedge \neg\texttt{At}^i(\texttt{b}, \texttt{ground})$$

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

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

Denote by A the collection of all such axioms.

Propositional logic for planning

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

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.

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

$$\begin{aligned} \texttt{Move}^0(\texttt{b}, \texttt{ground}, \texttt{spire}) &= \texttt{true} \\ \texttt{Move}^0(\texttt{a}, \texttt{spire}, \texttt{ground}) &= \texttt{true} \\ \boxed{\texttt{Move}^0(\texttt{a}, \texttt{ground}, \texttt{spire})} &= \texttt{true} \end{aligned}$$

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

We need a *precondition axiom*

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

and so on.

Propositional logic for planning

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

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

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

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

and so on.

These are *action-exclusion* axioms.

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

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

$$\neg(\texttt{At}^i(x, \texttt{l1}) \wedge \texttt{At}^i(x, \texttt{l2}))$$

for all combinations of x, i and $\texttt{l1} \neq \texttt{l2}$.

This is an example of a *state constraint*.

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

Review of constraint satisfaction problems (CSPs)

Recall that in a CSP we have:

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

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.

The state-variable representation

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

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

$$\mathscr{D}_1 = \{\texttt{climber1}, \texttt{climber2}\}$$
$$\mathscr{D}_2 = \{\texttt{home}, \texttt{jokeShop}, \texttt{hardwareStore}, \texttt{pavement}, \texttt{spire}, \texttt{hospital}\}$$
$$\mathscr{D}_3 = \{\texttt{rope}, \texttt{gorilla}\}$$

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

$$\mathscr{D}_1(\texttt{climber1})$$

and so on.

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

$$\texttt{above} \subseteq \mathscr{D}_1^{\texttt{above}} \times \mathscr{D}_2^{\texttt{above}}$$

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

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

The state-variable representation

The relation $\texttt{above}$ 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*

$$\texttt{at}(x_1, s) : \mathscr{D}_1^{\texttt{at}} \times S \to \mathscr{D}^{\texttt{at}}.$$

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

$$\texttt{sv}(x_1, \dots, x_n, s) : \mathscr{D}_1^{\texttt{sv}} \times \cdots \times \mathscr{D}_n^{\texttt{sv}} \times S \to \mathscr{D}^{\texttt{sv}}.$$

A state-variable denotes assertions such as

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

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:

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

The state-variable representation

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

For example:

Goal:
$\texttt{at}(\texttt{climber}, s) = \texttt{home}$ $\texttt{has}(\texttt{rope}, s) = \texttt{climber}$ $\texttt{at}(\texttt{gorilla}, s) = \texttt{spire}$

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

The state-variable representation

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

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

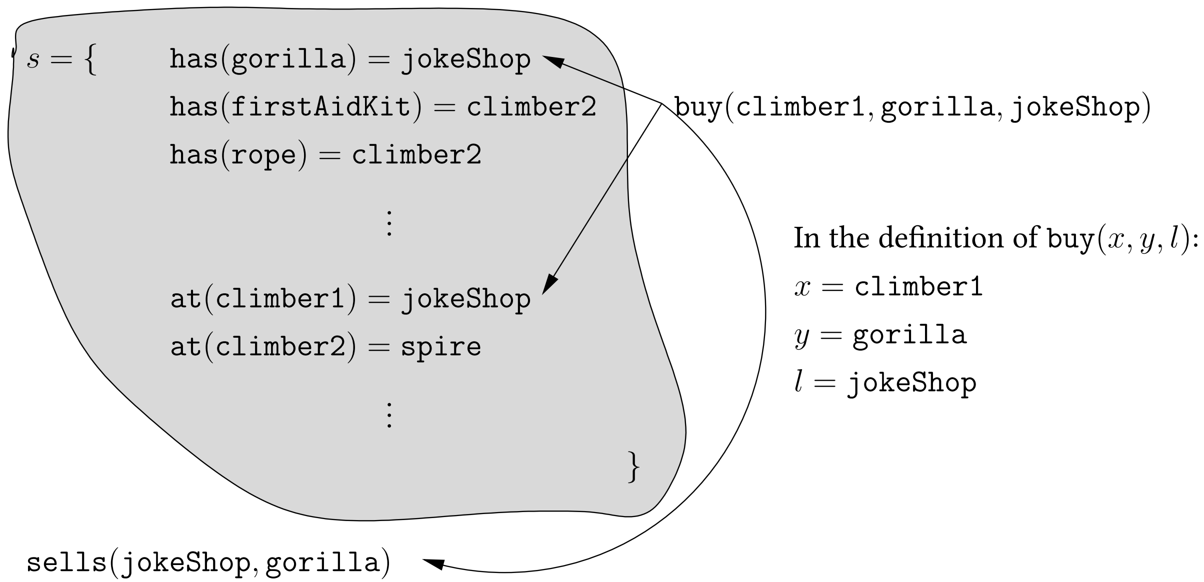
- For each state variable $\texttt{sv}$ consider all ground instances, such as $\texttt{sv}(\texttt{climber}, \texttt{rope})$, with arguments *consistent* with the *rigid relations*.
  Define X to be the set of all such ground instances.
- A state s is then just a set
  $$s = \{(v = c) | v \in X\}$$
  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.

The state-variable representation

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

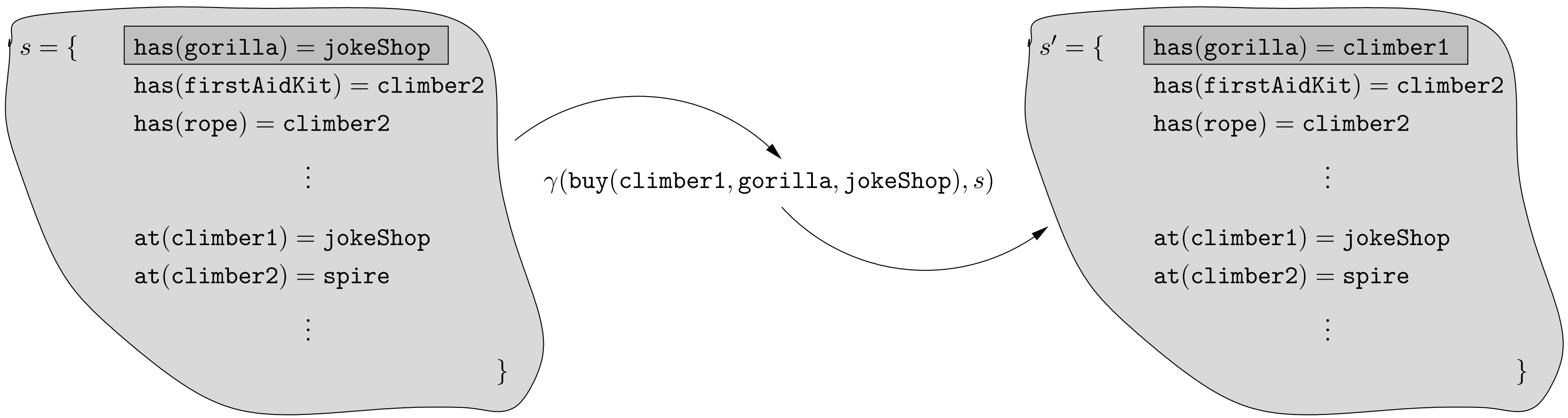


- An action is *applicable in* s if all expressions $v = c$ appearing in the set of preconditions also appear in s.
- As there is no rigid relation $sells$(jokeShop, fruitBats) we would *not* consider an action such as buy(climber1, fruitBats, jokeShop)—it is not *consistent with the rigid relations*.

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.*

The state-variable representation

A *solution* to a planning problem is a sequence $(a_0, a_1, \ldots, a_n)$ 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.

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

$$\texttt{action}^t$$

with domain

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

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

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

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

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

$$\texttt{sv}_i^t(c_1, \ldots, c_n)$$

with domain $D = \mathscr{D}^{\texttt{sv}_i}$. (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

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

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 $\texttt{a}(c_1, \ldots, c_n)$ with arguments *consistent with the rigid relations in its preconditions*:

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

$$(\texttt{action}^t = \texttt{a}(c_1, \ldots, c_n),\\ \texttt{sv}_i^t = v)$$

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

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

$$l = \texttt{jokeShop}$$

and

$$y = \texttt{gorilla}$$

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

Converting to a CSP

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

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 $\mathtt{a}(c_1, \ldots, c_n)$ with arguments *consistent with the rigid relations in its preconditions*:

For an effect of the form $\mathtt{sv}_i = v$ include constraint pairs

$$\left(\mathtt{action}^t = \mathtt{a}(c_1, \ldots, c_n), \\ \mathtt{sv}_i^{t+1} = v\right)$$

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

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

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 $\texttt{a}(c_1, \ldots, c_n)$ with arguments *consistent with the rigid relations in its preconditions*.
3. Each $\texttt{sv}_\texttt{i}$ that *does not appear in the effects of* $\texttt{a}$, and each $v \in \mathscr{D}^{\texttt{sv}_i}$

include in the CSP the ternary constraint

$$\begin{gathered}(\texttt{action}^t = \texttt{a}(c_1, \ldots, c_n), \\ \texttt{sv}_i^t = v, \\ \texttt{sv}_i^{t+1} = v).\end{gathered}$$

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 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 $\texttt{action}^t$ 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.