Principles of Intelligent Systems: Situation Calculus[*]

Lecture 11

[*]These slides are based on Chapters 8 and 10 of Russell and Norvig's *Artificial Intelligence: A modern approach* (http://aima.eecs.berkeley.edu/slides-pdf/)

Using first-order logic

We can build a knowledge base that uses first-order logic in the same way as previous described for propositional logic

However, instead of using TELL to add simple facts to the knowledge base we now add assertions that certain sentences are true, for example:

TELL$(KB, King(John))$.
TELL$(KB, \forall x\ King(x) \Rightarrow Person(x))$.

We can then ask questions of the knowledge base using ASK, for example:

ASK$(KB, King(John))$.

which returns $true$

Variable substitution

However, a major advance over propositional logic is that we can now ask **queries** that contain quantified variables

In this case the knowledge base should return all possible values for the variable in the query, by constructing a **substitution** or **binding** list

For example:

ASK$(KB, \exists x\ Person(x))$.

should return an answer $\{x/John\}$

The declarative programming language Prolog allows you to enter sentences as definite (Horn) clauses in first-order logic, and using its own inference engine, will automatically return query answers that contain all valid variable bindings

Numbers

First-order logic is powerful and expressive. As an example, we can construct a theory of natural numbers (nonnegative integers) using a single predicate ($NatNum$), a single constant (0) and a single successor function (S) producing the **Peano axioms** for natural numbers and addition:

$$
\begin{aligned}
&NatNum(0).\\
&\forall n \; NatNum(n) \Rightarrow NatNum(S(n)).
\end{aligned}
$$

meaning 0 is a natural number and for every object n, if n is a natural number, then $S(n)$ is a natural number. This is recursive, so the natural numbers are: $0, S(0), S(S(0)), \ldots$ As with all recursion S needs to be constrained, and then it can be used to define addition:

$$
\begin{aligned}
&\forall n \; 0 \neq S(n).\\
&\forall m, n \; m \neq n \Rightarrow S(m) \neq S(n).\\
&\forall n \; NatNum(n) \Rightarrow +(0, n) = n.\\
&\forall m, n \; NatNum(m) \wedge NatNum(n) \Rightarrow +(S(m), n) = S(+(m, n)).
\end{aligned}
$$

Lists

Lists can be easily expressed in first-order logic; as with the use of succession and 0 to express natural numbers, a list can be built up by adding single elements to an empty list:

By convention the empty list is expressed as $[\,]$ although it could equally well be expressed by a constant such as *Nil*

We can then define a function *Cons* that constructs a list, e.g. *Cons(x,y)* which can equivalently be written as $[x|y]$

$Cons$ represents a single list made up of two elements which can themselves be lists; to initialize a list we can add a single element to the empty list: $Cons(x, Nil)$ which can be equivalently expressed as $[x]$

A list of several elements, such as $[A, B, C]$ corresponds to the nested term: $Cons(A, Cons(B, Cons(C, Nil)))$

Wumpus world in first-order logic

Recall our propositional logic wumpus world problem
A wumpus world agent gets five possible on/off sensor readings:

$$Stench, Breeze, Glitter, Bump, Scream$$

The agent also needs to know the time step at which a percept occurred in order to reason correctly; Hence a possible percept for our agent expressed in first-order logic could be:

$$Percept([Stench, Breeze, Glitter, None, None], 5).$$

where 5 represents the time step in which the percept occurred
Such percept data trivially implies facts about the state at time t :

$$\forall t, s, g, m, c\ Percept([s, Breeze, g, m, c], t) \Rightarrow Breeze(t).$$
$$\forall t, s, b, m, c\ Percept([s, b, Glitter, m, c], t) \Rightarrow Glitter(t).$$

etc.

Actions in wumpus world

Actions in wumpus world can be represented by the following terms:

$$Turn(Right), Turn(Left), Forward, Shoot, Grab, Release, Climb$$

A first-order logic agent in wumpus world will want to know what action to take next; for the situation at time $t = 5$ this can be formulated as:

$$\exists a\ BestAction(a, 5).$$

Given a first-order knowledge-base that accurately models wumpus world, an ASK query should return an answer to this query in the form of a variable binding, such as $\{a/Grab\}$; the system should also TELL the knowledge-base that it is taking this action

Within this architecture we can now write general rules of behaviour:

$$\forall t\ Glitter(t) \Rightarrow BestAction(Grab, t).$$

Modelling the environment

First-order logic can also be used to express general properties of the environment, for instance the adjacency of any two squares can be evaluated using:

$$\forall x, y, a, b \; Adjacent([x, y], [a, b]) \Leftrightarrow \\ \quad [a, b] \in \{[x + 1, y], [x - 1, y], [x, y + 1], [x, y - 1]\}.$$

(here the set membership operator $\in$ would also need to be defined using another predicate)

Given a predicate $At(Agent, s, t)$, meaning the agent is at square s at time t, we can conclude a square is breezy regardless of the time period:

$$\forall s, t \; At(Agent, s, t) \wedge Breeze(t) \Rightarrow Breezy(s).$$

Diagnostic rules

Now we can define rules that go from observed effects to hidden causes, such as that a breezy square implies an adjacent pit:

$$\forall s\ Breezy(s) \Rightarrow \exists r\ Adjacent(r, s) \wedge Pit(r).$$

A system can only reason about what it has been told, hence we also need:

$$\forall s\ \neg Breezy(s) \Rightarrow \neg\exists r\ Adjacent(r, s) \wedge Pit(r).$$

These can be combined to obtain the bi-conditional sentence:

$$\forall s\ Breezy(s) \Leftrightarrow \exists r\ Adjacent(r, s) \wedge Pit(r).$$

Causal rules

Alternatively, the previous diagnostic rule can be expressed causally:

$$\forall r\ Pit(r) \Rightarrow [\forall s\ Adjacent(r,s) \Rightarrow Breezy(s)].$$
(a pit causes all adjacent squares to be breezy)

$$\forall s\ [\forall r\ Adjacent(r,s) \Rightarrow \neg Pit(r)] \Rightarrow \neg Breezy(s).$$
(if all adjacent squares are pitless, square will not be breezy)

Systems that reason with causal rules like this are known as model-based reasoning systems because causal rules provide a model of how the environment operates

In contrast, diagnostic systems reason from effects to possible causes, although these systems also contain an implicit model of how the environment operates - hence the above causal rules can be transformed into the diagnostic rule of the previous slide

Reasoning about time

Reasoning about time requires an explicit representation of the states of the world as time changes. We can no longer assume that everything stays the same - actions change some aspects of the world and not others

In the wumpus world our agent changes the environment by changing his position and may further change the environment by picking up the gold, firing his only arrow and/or killing the wumpus

Modelling such a world means we cannot be sure of the truth of many sentences until we have considered the actions that have previously occurred in the environment

For propositional logic time presents major problems: axioms must be copied and defined *for each time period*

Using first-order logic we can avoid this by quantifying over time, e.g.:
"$\forall t$, such-and-such is the result at $t+1$ of doing an action at time t"

Situation calculus

However, time can also be reasoned about in terms of situations, where one situation follows another as a result of an action; this provides a natural and expressive model of a changing environment without using explicit time intervals

Situation calculus is a formalism for expressing time in terms of situations and actions within the framework of first-order logic; situation calculus divides the world up into four categories:

– Actions: e.g. $Forward, Turn(Right)$
– Situations: given a starting situation S_0, the function $Result(a, s)$ names the situation that results when action a is executed in situation s
– Fluents: functions or predicates that can vary from one situation to the next
– Atemporal predicates: functions or predicates that *do not* vary between situations

Planning in situation calculus

Situation calculus allows us to reason about sequences of actions and therefore make plans, i.e. a sequence of events that will lead to a desired goal

To do this we can use the $Result(a, s)$ predicate to recursively collect the action sequence that can lead to a desired state

Firstly, we say that executing an empty sequence leaves the situation unchanged:

$$Result([\,], s) = s.$$

and executing a nonempty sequence is the same as executing the first action and then executing the rest in the resulting situation:

$$Result([a|seq], s) = Result(seq, Result(a, s)).$$

Note: from now on, unless otherwise stated, we will implicitly assume that all variables are universally quantified

Planning in wumpus world

Suppose the agent is at $[1,1]$ and the gold is at $[1,2]$; the goal is to have the gold at $[1,1]$; the fluent predicates are $At(o,x,s)$ and $Holding(o,s)$

The initial knowledge base could include:

$$At(Agent,[1,1],S_0) \wedge At(G_1,[1,2],S_0).$$

However, this is not enough, we have to specify what *isn't* true in a given situation as well, hence we need:

$$At(o,x,S_0) \Leftrightarrow [(o = Agent \wedge x = [1,1]) \vee (o = G_1 \wedge x = [1,2])].$$

We also need to state that G_1 is gold and that [1,1] and [1,2] are adjacent:

$$Gold(G_1) \wedge Adjacent([1,1],[1,2]) \wedge Adjacent([1,2],[1,1]).$$

The plan generating query would be: $\exists\ seq\ At(G_1,[1,1],Result(seq,S_0))$.

Representing actions

In situation calculus actions require at least two kinds of axiom:

- Possibility axioms which specify when it is possible to execute an action
- Effect axioms which specify what happens when an action is executed

The axioms have the following form:

POSSIBILITY AXIOM: $Preconditions \Rightarrow Poss(a, s)$.
EFFECT AXIOM: $Poss(a, s) \Rightarrow$ *Changes that result from taking action.*

In our simplified wumpus world the possibility axioms are:

$$At(Agent, x, s) \wedge Adjacent(x, y) \Rightarrow Poss(Go(x, y), s).$$
$$Gold(g) \wedge At(Agent, x, s) \wedge At(g, x, s) \Rightarrow Poss(Grab(g), s).$$
$$Holding(g, s) \Rightarrow Poss(Release(g), s).$$

The frame problem

Continuing the previous example, the effect axioms are:

$$Poss(Go(x,y),s) \Rightarrow At(Agent,y,Result(Go(x,y),s)).$$
$$Poss(Grab(g),s) \Rightarrow Holding(g,Result(Grab(g),s)).$$
$$Poss(Release(g),s) \Rightarrow \neg Holding(g,Result(Release(g),s)).$$

Now we can conclude that our agent can reach $[1,2]$ fromS_0:

$$At(Agent,[1,2],Result(Go([1,1],[1,2]),S_0)).$$

But can we conclude the gold can be grabbed in the new situation? i.e.:

$$At(G_1,[1,2],Result(Go([1,1],[1,2]),S_0)).$$

Unfortunately we can't conclude the gold (G_1) is still at $[1,2]$ after our first action - first-order logic requires a proof of this and doesn't know that everything unaffected by an effect axiom stays the same - instead we have to explicitly state everything that stays the same - this is the frame problem

Solving the frame problem

To solve the frame problem we have to rewrite the effect axioms as successor-state axioms - these axioms *completely* specify the conditions that make a particular fluent true, i.e. the fluent is true if and only if the right-hand side of the expression is true:

SUCCESSOR-STATE-AXIOM:
Action is possible $\Rightarrow$
(*Fluent is true in result state* $\Leftrightarrow$
Action's effect made it true $\lor$ *It was true before and action left it alone*).

Successor state axioms for wumpus world

Location axiom: the agent is at y after executing an action either if the action was possible and moves the agent to y or the agent was already at y and the action did not move him anywhere else:

$$Poss(a,s) \Rightarrow (At(Agent,y,Result(a,s)) \Leftrightarrow \\ a = Go(x,y) \vee (At(Agent,y,s) \wedge a \neq Go(y,z))).$$

Holding axiom: the agent is holding g after executing an action if the action was a grab of g and the grab is possible or if the agent was already holding g and the action does not release it:

$$Poss(a,s) \Rightarrow (Holding(g,Result(a,s)) \Leftrightarrow \\ a = Grab(g) \vee (Holding(g,s) \wedge a \neq Release(g))).$$

The ramification problem

However, there is nothing in the preceding axioms that informs the knowledge-base that, if the agent is holding the gold, and the agent moves, then the gold moves too - this is an example of the ramification problem

Ramifications are implicit effects that arise from actions; we tend to overlook them because they are so basic to our understanding of the world

In wumpus world we need an even more general successor-state location axiom which states that an object o is at y if the agent went to y and o is the agent or something he was holding; or if o was already at y and didn't go anywhere, with o being the agent or something he was holding:

$$\begin{aligned}
&Poss(a,s) \Rightarrow (At(o,y,Result(a,s)) \Leftrightarrow \\
&(a = Go(x,y) \wedge (o = Agent \vee Holding(o,s))) \vee \\
&(At(o,y,s) \wedge \neg(\exists z\, y \neq z \wedge a = Go(y,z) \wedge (o = Agent \vee Holding(o,s))))).
\end{aligned}$$

Summary

First-order logic contains universal quantifiers and variables allowing us to express general rules and relationships succinctly

Hence we can use diagnostic and causal rules to model the environment

Handling changes in the environment over time is more complex, but can be modelled in first-order logic using situation calculus

Situation calculus divides the environment into actions, situations, fluents and atemporal predicates

Describing actions in situation calculus requires stating what does and does not change as the result of an action producing the frame problem

The frame problem can be handled by creating successor-state axioms that define how each fluent evolves over time - these axioms also have to consider the secondary or implicit effects of actions known as the ramification problem