Artificial Intelligence

Knowledge representation and reasoning

We now look at how an agent might represent knowledge about its environment, and reason with this knowledge to achieve its goals.

Initially we'll represent and reason using first order logic (FOL). Aims:

• To show how FOL can be used to represent knowledge about an environment in the form of both background knowledge and knowledge derived from percepts.

• To show how this knowledge can be used to derive non-perceived knowledge about the environment using a theorem prover.

• To introduce the situation calculus and demonstrate its application in a simple environment as a means by which an agent can work out what to do next.

Using FOL in all its glory can be problematic.

Later we'll look at how some of the problems can be addressed using semantic networks, frames, inheritance and rules.

Reading: AIMA, chapters 7 to 10.

Knowledge representation and reasoning

Earlier in the course we looked at what an agent should be able to do.

It seems that all of us—and all intelligent agents—should use logical reasoning to help us interact successfully with the world.

Any intelligent agent should:

• Possess knowledge about the environment and about how its actions affect the environment.

• Use some form of logical reasoning to maintain its knowledge as percepts arrive.

• Use some form of logical reasoning to deduce actions to perform in order to achieve goals.

This raises some important questions:

• How do we describe the current state of the world?

• How do we infer from our percepts, knowledge of unseen parts of the world?

• How does the world change as time passes?

• How does the world stay the same as time passes? (The frame problem.)

• How do we know the effects of our actions? (The qualification and ramification problems.)

We'll now look at one way of answering some of these questions.

FOL (arguably?) seems to provide a good way in which to represent the required kinds of knowledge: it is expressive, concise, unambiguous, it can be adapted to different contexts, and it has an inference procedure, although a semidecidable one.

In addition is has a well-defined syntax and semantics.
Logic for knowledge representation

Problem: it’s quite easy to talk about things like set theory using FOL. For example, we can easily write axioms like

\[ \forall S . \forall S' . ((\forall x . (x \in S \iff x \in S')) \Rightarrow S = S') \]

But how would we go about representing the proposition that if you have a bucket of water and throw it at your friend they will get wet, have a bump on their head from being hit by a bucket, and the bucket will now be empty and dented?

More importantly, how could this be represented within a wider framework for reasoning about the world?

It’s time to introduce The Wumpus...

Wumpus world

The rules of Wumpus World:

• Unfortunately the cave contains a number of pits, which EVIL ROBOT can fall into. Eventually his batteries will fail, and that’s the end of him.
• The cave also contains the Wumpus, who is armed with state-of-the-art Evil Robot Obliteration Technology.
• The Wumpus itself knows where the pits are and never falls into one.

EVIL ROBOT can move around the cave at will and can perceive the following:

• In a position adjacent to the Wumpus, a stench is perceived. (Wumpuses are famed for their lack of personal hygiene.)
• In a position adjacent to a pit, a breeze is perceived.
• In the position where the gold is, a glitter is perceived.
• On trying to move into a wall, a bump is perceived.
• On killing the Wumpus a scream is perceived.

In addition, EVIL ROBOT has a single arrow, with which to try to kill the Wumpus.

“Adjacent” in the following does not include diagonals.
Wumpus world

So we have:

Percepts: stench, breeze, glitter, bump, scream.

Actions: forward, turnLeft, turnRight, grab, release, shoot, climb.

Of course, our aim now is not just to design an agent that can perform well in a single cave layout.

We want to design an agent that can usually perform well regardless of the layout of the cave.

Example: Prolog

You have by now learned a little about programming in Prolog. For example:

\[
\text{concat([], L, L).}\]
\[
\text{concat([H|T], L, [H|L2]) :- concat(T, L, L2).}\]

is a program to concatenate two lists. The query

\[
\text{concat([1, 2, 3], [4, 5], X).}\]

results in

\[
X = [1, 2, 3, 4, 5].\]

What’s happening here? Well, Prolog is just a more limited form of FOL so...

Logic for knowledge representation

The fundamental aim is to construct a knowledge base $KB$ containing a collection of statements about the world—expressed in FOL—such that useful things can be derived from it.

Our central aim is to generate sentences that are true, if the sentences in the KB are true.

This process is based on concepts familiar from your introductory logic courses:

- **Entailment:** $KB \models \alpha$ means that the KB entails $\alpha$.
- **Proof:** $KB \vdash \alpha$ means that $\alpha$ is derived from the KB using inference procedure $i$. If $i$ is sound then we have a proof.
- $i$ is sound if it can generate only entailed $\alpha$.
- $i$ is complete if it can find a proof for any entailed $\alpha$.

Example: Prolog

... we are in fact doing inference from a KB:

- The Prolog programme itself is the KB. It expresses some knowledge about lists.
- The query is expressed in such a way as to derive some new knowledge.

How does this relate to full FOL? First of all the list notation is nothing but syntactic sugar. It can be removed: we define a constant called empty and a function called cons.

Now $[1, 2, 3]$ just means

\[
\text{cons(1, cons(2, cons(3, empty)) \ldots )}\]

which is a term in FOL.

I will assume the use of the syntactic sugar for lists from now on.
The program when expressed in FOL, says

\[
\forall x. \text{concat(} \text{empty, } x, x) \land \\
\forall h, t, l_1, l_2. \text{concat}(t, l_1, l_2) \rightarrow \text{concat}(\text{cons}(h, t), l_1, \text{cons}(h, l_2))
\]

The rule is simple—given a Prolog program:

- Universally quantify all the unbound variables in each line of the program and ...
- ... form the conjunction of the results.

If the universally quantified lines are \(L_1, L_2, \ldots, L_n\) then the Prolog programme corresponds to the KB

\[
\text{KB} = L_1 \land L_2 \land \cdots \land L_n
\]

Now, what does the query mean?

Prolog differs from FOL in that, amongst other things:

- It restricts you to using Horn clauses.
- Its inference procedure is not a full-blown proof procedure.
- It does not deal with negation correctly.

However the central idea also works for full-blown theorem provers.

If you want to experiment, you can obtain Prover9 from

https://www.cs.unm.edu/~mccune/mace4/

We’ll see a brief example now, and a more extensive example of its use later, time permitting...

Expressed in Prover9, the above Prolog program and query look like this:

```prolog
set(prolog_style_variables).
% This is the translated Prolog program for list concatenation.
% Prover9 has its own syntactic sugar for lists.
formulas(assumptions).
  concat([], L, L).
  concat(T, L, L2) :-
    concat([H:T], L, [H:L2]).
end_of_list.
% This is the query.
formulas(goals).
  exists X concat([1, 2, 3], [4, 5], X).
end_of_list.
```

Note: it is assumed that unbound variables are universally quantified.
You can try to infer a proof using

```
prover9 -f file.in
```

and the result is (in addition to a lot of other information):

1 `concat(T,L,L2) -> concat([H:T],L,[H:L2])#label(nonclause). [assumption].`
2 `(exists X concat([1,2,3],[4,5],X)) #label(nonclause) #label(goal). [goal].`
3 `concat([],A,A). [assumption].`
4 `-concat(A,B,C) | concat([D:A],B,[D:C]). [clausify(1)].`
5 `concat([1,2,3],[4,5],A). [deny(2)].`
6 `concat([a,b],B,[a:b]). [ur(4,a,3,a)].`
7 `concat([2,3],[4,5],a). [resolve(5,a,4,b)].`
8 `concat([a,b],C,[a,b:C]). [ur(4,a,5,a)].`
9 `F. [resolve(8,a,7,a)].`

This shows that a proof is found but doesn’t explicitly give a value for X—we’ll see how to extract that later...

---

The fundamental idea

So the basic idea is: build a KB that encodes knowledge about the world, the effects of actions and so on.

The KB is a conjunction of pieces of knowledge, such that:

- A query regarding what our agent should do can be posed in the form

  `∃actionList. Goal(...)actionList...`

- Proving that

  `KB → ∃actionList. Goal(...)actionList...`

instantiates actionList to an actual list of actions that will achieve a goal represented by the Goal predicate.

We sometimes use the notation `ask` and `tell` to refer to querying and adding to the KB.

---

Using FOL in AI: the triumphant return of the Wumpus

We want to be able to speculate about the past and about possible futures. So:

- We include situations in the logical language used by our KB.
- We include axioms in our KB that relate to situations.

This gives rise to situation calculus.

---

Situation calculus

In situation calculus:

- The world consists of sequences of situations.
- Over time, an agent moves from one situation to another.
- Situations are changed as a result of actions.

In Wumpus World the actions are: forward, shoot, grab, climb, release, turnRight, turnLeft.

- A situation argument is added to items that can change over time. For example

  `At(location, s)`

Items that can change over time are called fluents.

- A situation argument is not needed for things that don’t change. These are sometimes referred to as eternal or atemporal.
Representing change as a result of actions

Situation calculus uses a function
\[ \text{result}(\text{action}, s) \]
to denote the new situation arising as a result of performing the specified action in the specified situation.
\[ \text{result}(\text{grab}, s_0) = s_1 \]
\[ \text{result}(\text{turnLeft}, s_1) = s_2 \]
\[ \text{result}(\text{shoot}, s_2) = s_3 \]
\[ \text{result}(\text{forward}, s_3) = s_4 \]

Axioms I: possibility axioms

The first kind of axiom we need in a KB specifies when particular actions are possible.
We introduce a predicate
\[ \text{Poss}(\text{action}, s) \]
denoting that an action can be performed in situation \( s \).
We then need a possibility axiom for each action. For example:
\[ \text{At}(l, s) \land \text{Available}(\text{gold}, l, s) \rightarrow \text{Poss}(\text{grab}, s) \]
Remember that unbound variables are universally quantified.

Axioms II: effect axioms

Given that an action results in a new situation, we can introduce effect axioms to specify the properties of the new situation.
For example, to keep track of whether EVIL ROBOT has the gold we need effect axioms to describe the effect of picking it up:
\[ \text{Poss}(\text{grab}, s) \rightarrow \text{Have}(\text{gold}, \text{result}(\text{grab}, s)) \]
Effect axioms describe the way in which the world changes.
We would probably also include
\[ \neg\text{Have}(\text{gold}, s_0) \]
in the KB, where \( s_0 \) is the starting situation.

Important: we are describing what is true in the situation that results from performing an action in a given situation.

Axioms III: frame axioms

We need frame axioms to describe the way in which the world stays the same.
Example:
\[ \text{Have}(o, s) \land \neg \left( a = \text{release} \land a = \text{gold} \right) \land \neg \left( a = \text{shoot} \land a = \text{arrow} \right) \]
\[ \rightarrow \text{Have}(o, \text{result}(a, s)) \]
describes the effect of having something and not discarding it.
In a more general setting such an axiom might well look different. For example
\[ \neg\text{Have}(o, s) \land \neg \left( o \neq \text{grab}(o) \lor \neg\left( \text{Available}(o, s) \land \text{Portable}(o) \right) \right) \]
\[ \rightarrow \neg\text{Have}(o, \text{result}(a, s)) \]
describes the effect of not having something and not picking it up.
The frame problem

The frame problem has historically been a major issue.

Representational frame problem: a large number of frame axioms are required to represent the many things in the world which will not change as the result of an action.

We will see how to solve this in a moment.

Inferential frame problem: when reasoning about a sequence of situations, all the unchanged properties still need to be carried through all the steps.

This can be alleviated using planning systems that allow us to reason efficiently when actions change only a small part of the world. There are also other remedies, which we will not cover.

Successor-state axioms

Effect axioms and frame axioms can be combined into successor-state axioms. One is needed for each predicate that can change over time.

\[ \text{Poss}(a, s) \rightarrow \]
\[ (\text{Have}(o, \text{result}(a, s)) \leftrightarrow ((a = \text{grab} \land \text{Available}(o, s)) \lor \]
\[ (\text{Have}(o, s) \land \neg(a = \text{release} \land o = \text{gold}) \land \]
\[ \neg(a = \text{shoot} \land o = \text{arrow})))) \]

Knowing where you are, and so on...

We now have considerable flexibility in adding further rules:

- If \( s_0 \) is the initial situation we know that \( \text{At}((1, 1), s_0) \).
- We need to keep track of what way we’re facing. Say north is 0, south is 2, east is 1 and west is 3. We might assume \( \text{facing}(s_0) = 0 \).
- We need to know how motion affects location
  \[ \text{forwardResult}((x, y), \text{north}) = (x, y + 1) \]
  \[ \text{forwardResult}((x, y), \text{east}) = (x + 1, y) \]
  and so on.
- The concept of adjacency is very important in the Wumpus world
  \[ \text{Adjacent}(l_1, l_2) \iff \exists d \text{ forwardResult}(l_1, d) = l_2 \]
- We also know that the cave is 4 by 4 and surrounded by walls
  \[ \text{WallHere}((x, y)) \iff (x = 0 \lor y = 0 \lor x = 5 \lor y = 5) \]

The qualification and ramification problems

Qualification problem: we are in general never completely certain what conditions are required for an action to be effective.

Consider for example turning the key to start your car.

This will lead to problems if important conditions are omitted from axioms.

Ramification problem: actions tend to have implicit consequences that are large in number.

For example, if I pick up a sandwich in a dodgy sandwich shop, I will also be picking up all the bugs that live in it. I don’t want to model this explicitly.
Solving the ramification problem

The ramification problem can be solved by modifying successor-state axioms. For example:

\[
\text{Poss}(a, s) \rightarrow \left( \text{At}(o, l, \text{result}(a, s)) \iff \exists l'. \ a = \text{go}(l', l) \land [o = \text{robot} \lor \text{Has}(\text{robot}, o, s)] \lor \text{At}(o, l, s) \land \neg \exists l''. \ a = \text{go}(l, l'') \land 1 \neq l'' \land [o = \text{robot} \lor \text{Has}(\text{robot}, o, s)]) \right)
\]

This describes the fact that anything EVIL ROBOT is carrying moves around with him.

General axioms for situations and objects

*Note:* in FOL, if we have two constants robot and gold then an interpretation is free to assign them to be the same thing. This is not something we want to allow. Unique names axioms state that each pair of distinct items in our model of the world must be different:

- robot \(\neq\) gold
- robot \(\neq\) arrow
- robot \(\neq\) wumpus
- ...

Unique actions axioms state that actions must share this property, so for each pair of actions:

- \(\text{go}(l, l') \neq \text{grab}\)
- \(\text{go}(l, l') \neq \text{drop}(o)\)

and in addition we need to define equality for actions, so for each action:

- \(\text{go}(l, l') = \text{go}(l'', l''') \iff l = l'' \land l' = l'''\)
- \(\text{drop}(o) = \text{drop}(o') \iff o = o'\)

Deducing properties of the world: causal and diagnostic rules

If you know where you are, then you can think about places. Synchronic rules relate properties shared by a single state of the world. There are two kinds: causal and diagnostic.

**Causal rules**: some properties of the world will produce percepts.

- \(\text{WumpusAt}(l_1) \land \text{Adjacent}(l_1, l_2) \rightarrow \text{StenchAt}(l_2)\)
- \(\text{PitAt}(l_1) \land \text{Adjacent}(l_1, l_2) \rightarrow \text{BreezeAt}(l_2)\)

Systems reasoning with such rules are known as model-based reasoning systems. **Diagnostic rules**: infer properties of the world from percepts. For example:

- \(\text{At}(l, s) \land \text{Breeze}(s) \rightarrow \text{BreezeAt}(l)\)
- \(\text{At}(l, s) \land \text{Stench}(s) \rightarrow \text{StenchAt}(l)\)

These may not be very strong.

The difference between model-based and diagnostic reasoning can be important. For example, medical diagnosis can be done based on symptoms or based on a model of disease.

General axioms for situations and objects

The situations are ordered so:

\(s_0 \neq \text{result}(a, s)\)

and situations are distinct so:

\(\text{result}(a, s) = \text{result}(a', s') \iff a = a' \land s = s'\)

Strictly speaking we should be using a many-sorted version of FOL.

In such a system variables can be divided into sorts which are implicitly separate from one another.

Finally, we’re going to need to specify what’s true in the start state.

For example:

\[\text{At}(\text{robot}, [1, 1], s_0)\]
\[\text{At}(\text{wumpus}, [3, 4], s_0)\]
\[\text{Has}(\text{robot}, \text{arrow}, s_0)\]

and so on.
Sequences of situations

We know that the function \texttt{result} tells us about the situation resulting from performing an action in an earlier situation.

How can this help us find \textit{sequences of actions to get things done}?

Define

\begin{align*}
\text{Sequence}([], s, s') &= s' = s \\
\text{Sequence}([a], s, s') &= \text{Poss}(a, s) \land s' = \text{result}(a, s) \\
\text{Sequence}(a :: as, s, s') &= \exists t. \text{Sequence}([a], s, t) \land \text{Sequence}(as, t, s')
\end{align*}

To obtain a \textit{sequence of actions that achieves} \texttt{Goal}(s) we can use the query

\[ \exists a \exists s. \text{Sequence}(a, s_0, s) \land \text{Goal}(s) \]

Interesting reading

Knowledge representation based on logic is a vast subject and can’t be covered in full in the lectures.

In particular:

- Techniques for representing \textit{further kinds of knowledge}.
- Techniques for moving beyond the idea of a \textit{situation}.
- Reasoning systems based on \textit{categories}.
- Reasoning systems using \textit{default information}.
- \textit{Truth maintenance systems}.

Happy reading...