Artificial Intelligence

Knowledge representation and reasoning

Reading: AIMA, chapters 7 to 10.

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*.

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

$$\forall S \,.\, \forall S' \,.\, ((\forall x \,.\, (x \in S \Leftrightarrow x \in S')) \Rightarrow S = S')$$

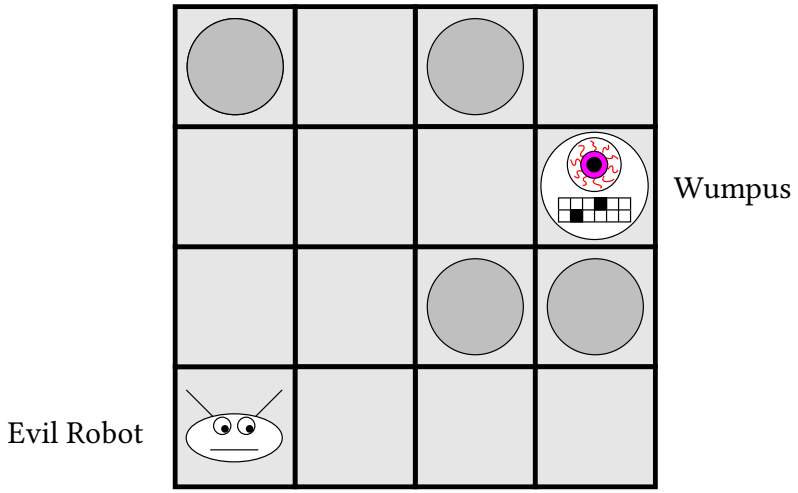
But how would we go about representing the proposition that *if you have a bucket of water and throw it at your friend they will get wet, have a bump on their head from being hit by a bucket, and the bucket will now be empty and dented*?

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

As a simple test scenario for a knowledge-based agent we will make use of the *Wumpus World*.



The Wumpus World is a 4 by 4 grid-based cave.

EVIL ROBOT wants to enter the cave, find some gold, and get out again unscathed.

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.

Wumpus world

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.

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: $\texttt{KB} \models \alpha$ means that the KB entails α.
- Proof: $\texttt{KB} \vdash_i \alpha$ means that α 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 α.
- i is *complete* if it can find a proof for *any* entailed α.

Example: Prolog

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

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

is a program to concatenate two lists. The query

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

results in

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

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

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

$$\texttt{cons}(1, \texttt{cons}(2, \texttt{cons}(3, \texttt{empty})))$$

which is a term in FOL.

I will assume the use of the syntactic sugar for lists from now on.

Prolog and FOL

The program when expressed in FOL, says

$$\forall x \,.\, \texttt{concat}(\texttt{empty}, x, x) \wedge$$
$$\forall h, t, l_1, l_2 \,.\, \texttt{concat}(t, l_1, l_2) \rightarrow \texttt{concat}(\texttt{cons}(h, t), l_1, \texttt{cons}(h, l_2))$$

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

$$\texttt{KB} = L_1 \wedge L_2 \wedge \cdots \wedge L_n$$

Now, what does the query mean?

Prolog and FOL

When you give the query

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

to Prolog it responds by *trying to prove* the following statement

$$\texttt{KB} \rightarrow \exists X \,.\, \texttt{concat}([1,2,3],[4,5],X)$$

So: it tries to prove that the KB *implies the query*, and variables in the query are existentially quantified.

When a proof is found, it supplies a *value for* X that *makes the inference true*.

Prolog and FOL

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...

Prolog and FOL

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

```
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(non_clause).  [assumption].
2 (exists X concat([1,2,3],[4,5],X)) # label(non_clause) # 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,[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,6,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*

$$\exists \texttt{actionList} \,.\, \texttt{Goal}(\ldots\texttt{actionList}\ldots)$$

- Proving that

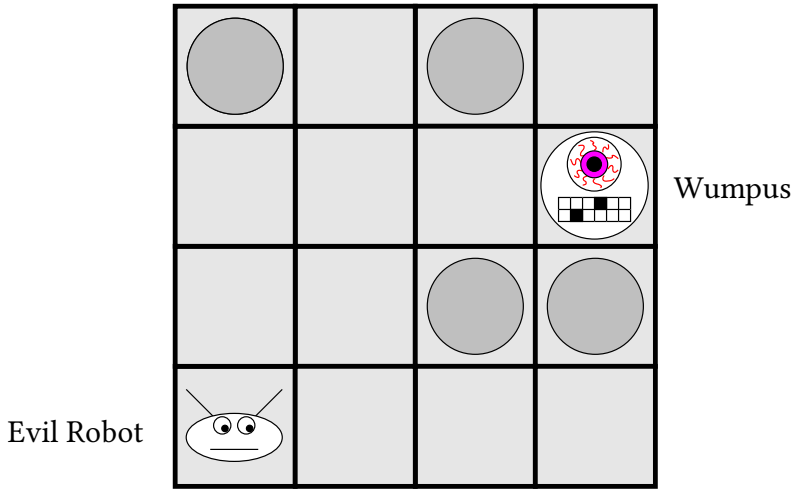
$$\texttt{KB} \rightarrow \exists \texttt{actionList} \,.\, \texttt{Goal}(\ldots\texttt{actionList}\ldots)$$

  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

  $$\texttt{At}(\texttt{location}, s)$$

  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

$$\texttt{result}(\texttt{action}, s)$$

to denote the *new* situation arising as a result of performing the specified action in the specified situation.

$$\begin{aligned}
\texttt{result}(\texttt{grab}, s_0) &= s_1 \\
\texttt{result}(\texttt{turnLeft}, s_1) &= s_2 \\
\texttt{result}(\texttt{shoot}, s_2) &= s_3 \\
\texttt{result}(\texttt{forward}, s_3) &= s_4 \\
&\vdots
\end{aligned}$$

Axioms I: possibility axioms

The first kind of axiom we need in a KB specifies *when particular actions are possible*.

We introduce a predicate

$$\texttt{Poss}(\texttt{action}, s)$$

denoting that an action can be performed in situation s.

We then need a *possibility axiom* for each action. For example:

$$\texttt{At}(l, s) \land \texttt{Available}(\texttt{gold}, l, s) \rightarrow \texttt{Poss}(\texttt{grab}, s)$$

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:

$$\texttt{Poss}(\texttt{grab}, s) \rightarrow \texttt{Have}(\texttt{gold}, \texttt{result}(\texttt{grab}, s))$$

Effect axioms describe the way in which the world *changes*.

We would probably also include

$$\neg\texttt{Have}(\texttt{gold}, s_0)$$

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:

$$\begin{aligned}&\texttt{Have}(o, s) \land \\ &\quad \neg(a = \texttt{release} \land o = \texttt{gold}) \land \neg(a = \texttt{shoot} \land o = \texttt{arrow}) \\ &\quad \rightarrow \texttt{Have}(o, \texttt{result}(a, s))\end{aligned}$$

describes the effect of *having something and not discarding it*.

In a more general setting such an axiom might well look different. For example

$$\begin{aligned}&\neg\texttt{Have}(o, s) \land \\ &\quad (a \neq \texttt{grab}(o) \lor \neg(\texttt{Available}(o, s) \land \texttt{Portable}(o))) \\ &\quad \rightarrow \neg\texttt{Have}(o, \texttt{result}(a, s))\end{aligned}$$

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.

Action a is possible $\rightarrow$
(true in new situation $\iff$
(you did something to make it true $\vee$
it was already true and you didn't make it false))

For example

$$\texttt{Poss}(a, s) \rightarrow$$
$$(\texttt{Have}(o, \texttt{result}(a, s)) \iff ((a = \texttt{grab} \wedge \texttt{Available}(o, s)) \vee$$
$$(\texttt{Have}(o, s) \wedge \neg(a = \texttt{release} \wedge o = \texttt{gold}) \wedge$$
$$\neg(a = \texttt{shoot} \wedge o = \texttt{arrow}))))$$

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 $\texttt{At}((1,1), s_0)$.
- 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 $\texttt{facing}(s_0) = 0$.
- We need to know how motion affects location

$$\texttt{forwardResult}((x,y), \texttt{north}) = (x, y+1)$$

$$\texttt{forwardResult}((x,y), \texttt{east}) = (x+1, y)$$

  and so on.
- The concept of adjacency is very important in the Wumpus world

$$\texttt{Adjacent}(l_1, l_2) \iff \exists d\, \texttt{forwardResult}(l_1, d) = l_2$$

- We also know that the cave is 4 by 4 and surrounded by walls

$$\texttt{WallHere}((x,y)) \iff (x = 0 \vee y = 0 \vee x = 5 \vee y = 5)$$

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:

$$
\begin{aligned}
&\texttt{Poss}(a, s) \rightarrow \\
&\quad (\texttt{At}(o, l, \texttt{result}(a, s)) \iff \\
&\quad (\exists l' \,.\, a = \texttt{go}(l', l) \wedge \\
&\qquad\qquad [o = \texttt{robot} \vee \texttt{Has}(\texttt{robot}, o, s)]) \vee \\
&\qquad (\texttt{At}(o, l, s) \wedge \\
&\qquad\qquad [\neg \exists l'' \,.\, a = \texttt{go}(l, l'') \wedge l \neq l'' \wedge \\
&\qquad\qquad \{o = \texttt{robot} \vee \texttt{Has}(\texttt{robot}, o, s)\}]))
\end{aligned}
$$

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

Deducing properties of the world: causal and diagnostic rules

If you know where you are, then you can think about *places* rather than just *situations*. *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.

$$\texttt{WumpusAt}(l_1) \land \texttt{Adjacent}(l_1, l_2) \rightarrow \texttt{StenchAt}(l_2)$$

$$\texttt{PitAt}(l_1) \land \texttt{Adjacent}(l_1, l_2) \rightarrow \texttt{BreezeAt}(l_2)$$

Systems reasoning with such rules are known as *model-based* reasoning systems.

Diagnostic rules: infer properties of the world from percepts. For example:

$$\texttt{At}(l, s) \land \texttt{Breeze}(s) \rightarrow \texttt{BreezeAt}(l)$$

$$\texttt{At}(l, s) \land \texttt{Stench}(s) \rightarrow \texttt{StenchAt}(l)$$

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

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

$$\begin{aligned} \texttt{robot} &\neq \texttt{gold} \\ \texttt{robot} &\neq \texttt{arrow} \\ \texttt{robot} &\neq \texttt{wumpus} \\ &\vdots \end{aligned}$$

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

$$\begin{aligned} \texttt{go}(l, l') &\neq \texttt{grab} \\ \texttt{go}(l, l') &\neq \texttt{drop}(o) \\ &\vdots \end{aligned}$$

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

$$\begin{aligned} \texttt{go}(l, l') = \texttt{go}(l'', l''') &\iff l = l'' \wedge l' = l''' \\ \texttt{drop}(o) = \texttt{drop}(o') &\iff o = o' \\ &\vdots \end{aligned}$$

General axioms for situations and objects

The situations are *ordered* so

$$s_0 \neq \texttt{result}(a, s)$$

and situations are *distinct* so

$$\texttt{result}(a, s) = \texttt{result}(a', s') \iff a = a' \wedge s = s'$$

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

$$\begin{gathered}\texttt{At}(\texttt{robot}, [1, 1], s_0) \\ \texttt{At}(\texttt{wumpus}, [3, 4], s_0) \\ \texttt{Has}(\texttt{robot}, \texttt{arrow}, s_0) \\ \vdots\end{gathered}$$

and so on.

Sequences of situations

We know that the function `result` tells us about the situation resulting from performing an action in an earlier situation.

How can this help us find *sequences of actions to get things done*?

Define

$$\begin{aligned}
\texttt{Sequence}([], s, s') &= s' = s \\
\texttt{Sequence}([a], s, s') &= \texttt{Poss}(a, s) \wedge s' = \texttt{result}(a, s) \\
\texttt{Sequence}(a :: as, s, s') &= \exists t \,.\, \texttt{Sequence}([a], s, t) \wedge \texttt{Sequence}(as, t, s')
\end{aligned}$$

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

$$\exists a \; \exists s \,.\, \texttt{Sequence}(a, s_0, s) \wedge \texttt{Goal}(s)$$

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

Happy reading…