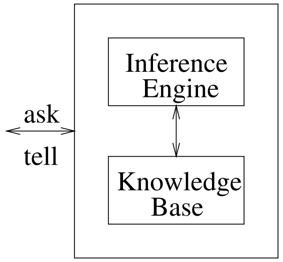
Knowledge Bases and Knowledge Engineering

1 Knowledge Bases

Our agent needs to represent and reason with large quantities of knowledge about the world. This knowledge is kept in a *knowledge base*. This contains a set of statements about the world, expressed in some logic/knowledge representation language. Reasoning with this knowledge is done by an *inference engine*. An inference engine is a program that draws conclusions from the knowledge in the knowledge base; it will implement one of the logic's *proof theories*.

Any system that contains both a knowledge base and an inference engine may be called a *knowledge-based system* (KBS). Such a system will offer (at least) two methods in its public interface: `tell` and `ask`. `tell` is used to insert a new wff into the knowledge base; `ask` is used to query the knowledge base.



Knowledge-Based System

A *database system* also has methods that are seemingly equivalent to `tell` and `ask` (e.g. `INSERT` and `SELECT` in SQL respectively). While there are no clear-cut differences between KBS and database systems, it may help us to understand KBS better if we try to draw some distinctions:

- One difference is in the expressiveness of what can be `tell`ed. A typical relational database, for example, can only store data as rows in tables. Logically, it can only store wffs that are atoms whose arguments are all constant symbols. A KBS is less restrictive in the wffs it can store.
- Perhaps more importantly, there is a difference in the way that database systems and KBS respond to queries that they have been `ask`ed. In both cases, the answers must follow from what has been `tell`ed, but, in the case of KBS, extended chains of reasoning might be used to answer the queries. The answer may only be implicit in what was `tell`ed.

(One should note at this point that there is a large body of research into systems called 'deductive databases'. These systems blur the distinction between database systems and KBS.)

2 Knowledge Engineering

A *domain* is some aspect of the world. Many current AI agents are domain-specific: they focus on one (or a very small number) of aspects of the world, e.g. meningitis diagnosis.

We refer to the process of building a knowledge base for one or more domains as *knowledge engineering*. It is carried out by AI experts, *knowledge engineers*, in conjunction with people who are familiar with the domain, *domain experts* (e.g. medical consultants). The knowledge engineers must become acquainted with the domain, usually by interviewing the domain experts, and this part of knowledge engineering is known as *knowledge elicitation* or *knowledge acquisition*. Then, formal representations can be written and placed into the knowledge base.

The process of knowledge engineering involves such steps as:

Ontological engineering

Decide what to talk about. The knowledge engineer must first determine what is relevant and irrelevant to the system s/he wishes to build. What kinds of things exist in this domain? E.g. is time relevant? Are events, states and processes relevant or are we interested only in physical objects? Which physical objects are we interested in? Are all objects atomic or can they have subparts? Etc.

Decide on a vocabulary of predicate symbols, function symbols and constant symbols. For example, should some relationship be represented as a function symbol or as a predicate symbol? For example, to represent *"The Sun is yellow"*, we could choose between

- *yellow*(*sun*),
- *colour*(*sun*, *yellow*)
- *propertyOf*(*sun*, *colour*, *yellow*)
- *equals*(*colourOf*(*sun*), *yellow*)

and any number of further options.

Class exercise. Give some advantages/disadvantages for these options.

Axiomatisation

Encode general knowledge. Write wffs to capture relationships between concepts.

Encode specific knowledge. Write wffs to capture specific instances.

We'll look at an example that uses these four steps in the next lecture. But, in the current lecture, we'll take a much simpler example domain so that we can practice writing wffs.

3 An Example Domain

Before we start on the example, let's check our ability to convert two simple English sentences into FOPL.

Class exercise. Which are the correct translations into FOPL?

- *Every person is happy.*
$$\forall x(\mathit{person}(x) \land \mathit{happy}(x))$$
$$\forall x(\mathit{person}(x) \Rightarrow \mathit{happy}(x))$$
- *Some person is happy.*
$$\exists x(\mathit{person}(x) \land \mathit{happy}(x))$$
$$\exists x(\mathit{person}(x) \Rightarrow \mathit{happy}(x))$$

Now we'll encode a few facts about the domain of family relationships. We'll use the following 'key' for the unary predicate symbols *male* and *female*, the binary predicate symbols *parent*, *child*, *grandparent*, *sibling* and *equals*, and the unary function symbols *father* and *mother*:

$\mathit{male}(x)$	:	x is male
$\mathit{female}(x)$	:	x is female
$\mathit{parent}(x, y)$	:	x is a parent of y
$\mathit{child}(x, y)$	:	x is a child of y
$\mathit{grandparent}(x, y)$	:	x is a grandparent of y
$\mathit{sibling}(x, y)$	:	x is a sibling of y
$\mathit{equals}(x, y)$	:	x is equal to y
$\mathit{father}(x)$	:	the father of x
$\mathit{mother}(x)$	:	the mother of x

(For tidiness, we'll allow ourselves to write $x = y$ using $=$ as an infix predicate symbol in place of the more cumbersome and less familiar but syntactically more accurate $\mathit{equals}(x, y)$.)

Here are some facts we'll translate into logic.

- Male and female are disjoint sets.
- Parent and child are inverse relationships.
- A grandparent is a parent of one's parent.
- One's mother is one's female parent.
- A sibling is another child of one's parents.

And so on!

In fact, strictly we can't just use *equals* (or $=$) without also tackling the domain of equality. We need to write wffs to capture the following facts:

- Equality is reflexive
$$\forall x\,(x = x)$$
- Equality is symmetric
$$\forall x \forall y (x = y \Rightarrow y = x)$$
- Equality is transitive
$$\forall x \forall y \forall z ((x = y \land y = z) \Rightarrow x = z)$$
- Equality is substitutive
$$\forall x \forall y ((\mathit{male}(x) \land x = y) \Rightarrow \mathit{male}(y))$$
$$\forall x \forall y ((\mathit{female}(x) \land x = y) \Rightarrow \mathit{female}(y))$$
  and so on for all other predicate symbols and function symbols

(There is an alternative to doing this, and this is to revise our semantics for FOPL. Within a revised semantics, we can give the *equals* predicate symbol a special, fixed interpretation. This would remove the need to axiomatise the reflexivity, symmetry, transitivity and substitutivity properties.)

4 What is a good set of wffs?

The statements that are in the knowledge base initially are sometimes called *axioms*. They are our basic facts about the domain. It is from these that the inference engine derives other facts (sometimes called *theorems*). The question is: how do we know when we've got a good set of axioms in our knowledge base?

Well, obviously, we need to write *'enough'* wffs. We want to keep writing axioms until all true facts about our domain (that we deem relevant) follow from the axioms, and only facts that are true in the domain follow from the axioms.

But mathematicians often try to write a *minimal* set of axioms. They try to make it the case that they never write redundant axioms. A redundant axiom would be one that could be derived from the other axioms. They aim for a minimal set from which all other facts about the domain (that they deem relevant) can be derived.

Some knowledge engineers strive for similar elegance when representing knowledge about a domain. However, this is not strictly necessary: our goal is not elegance. Efficiency of reasoning is often much more important. And here the issue becomes less clear-cut. On the one hand, the more facts about the domain that we represent explicitly (including redundant ones), then the less reasoning we need to do, which should improve efficiency. On the other hand, the more axioms in the knowledge base (including redundant ones), then the more 'stuff' there is that the inference engine has to plough through, which may worsen efficiency. Some compromise is needed.

Exercise

In addition to the predicate symbols and function symbols used earlier, we'll now also use the following:

$\mathit{brother}(x, y)$	:	x is a brother of y
$\mathit{sister}(x, y)$	:	x is a sister of y
$\mathit{aunt}(x, y)$	:	x is an aunt of y
$\mathit{uncle}(x, y)$	:	x is an uncle of y
$\mathit{cousin}(x, y)$	:	x is a (first) cousin of y
$\mathit{ancestor}(x, y)$	:	x is an ancestor of y

Now translate the following statements into FOPL:

1. Your brother is your male sibling.
2. Your sister is your female sibling.
3. An aunt is a sister of a parent.
4. An uncle is a brother of a parent.
5. Cousins are children of aunts or uncles.
6. One's ancestors are one's parents or ancestor's of one's parents. (A recursive definition!)