COMP9016 Knowledge Representation

Knowledge Representation

Dr Ruairi O'Reilly

October 22, 2019

Cork Institute of Technology

Outline

Ontological Engineering

- Toy domains vrs Complex domains?
- Examples of Complex domains (shopping on the Internet or driving a car in traffic) $\longrightarrow$ more general and flexible representations.
- How to create these representations, concentrating on general concepts—such as Events, Time, Physical Objects, and Beliefs— that occur in many different domains. Representing these abstract concepts is sometimes called ontological engineering.
- Representing everything in the world …

Upper Ontology

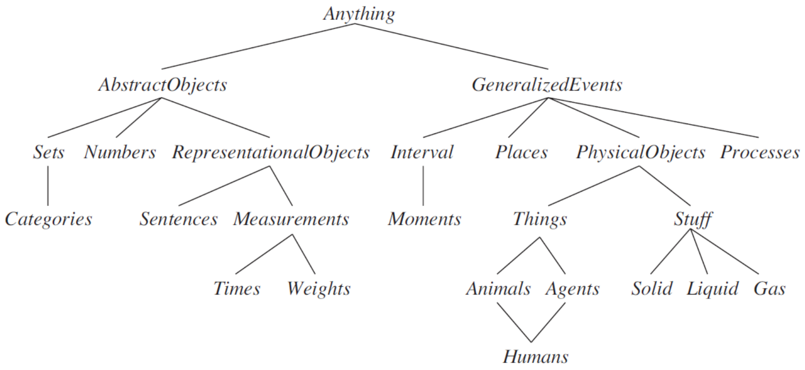

Figure 12.1 The upper ontology of the world, showing the topics to be covered later in the chapter. Each link indicates that the lower concept is a specialization of the upper one. Specializations are not necessarily disjoint; a human is both an animal and an agent, for example. We will see in Section 12.3.3 why physical objects come under generalized events.

Figure 1: Upper Ontology - what about exceptions?

Use of an upper ontology? Wumpus World

Although we do represent time, it has a simple structure - nothing happens except when the agent acts, and all changes are instantaneous.

- A more general ontology, better suited for the real world, $\longrightarrow$ would allow for simultaneous changes extended over time.
- We also used a Pit predicate to say which squares have pits $\longrightarrow$ We could have allowed for different kinds of pits by having several individuals belonging to the class of pits, each having different properties.
- Similarly, we might want to allow for other animals besides wumpuses $\longrightarrow$ It might not be possible to pin down the exact species from the available percepts, so we would need to build up a biological taxonomy to help the agent predict the behavior of cave-dwellers from scanty clues.

General Purpose Ontology

Two major characteristics of general-purpose ontologies distinguish them from collections of special-purpose ontologies:

- A general-purpose ontology should be applicable in more or less any special-purpose domain (with the addition of domain-specific axioms). This means that no representational issue can be finessed or brushed under the carpet.
- In any sufficiently demanding domain, different areas of knowledge must be unified, because reasoning and problem solving could involve several areas simultaneously.

Note: None of the top AI applications make use of a shared ontology—they all use special-purpose knowledge engineering. Agree on an Ontology?

Categories And Objects

Categories And Objects

Choices for representing categories in first-order logic - predicates and objects `Basketball (b)`, or we can reify the category as an object, `Basketballs`. We could then say `Member(b, Basketballs)`, which we will abbreviate as $\mathtt{b} \in \mathtt{Basketballs}$, to say that `b` is a member of the category of `basketballs`. We say `Subset(Basketballs, Balls)`, abbreviated as $\mathtt{Basketballs} \subset \mathtt{Balls}$, to say that `Basketballs` is a subcategory of `Balls`. We will use subcategory, subclass, and subset interchangeably.

Categories serve to organize and simplify the knowledge base through inheritance.

Subclass relations organize categories into a taxonomy, or taxonomic hierarchy. Taxonomies have been used explicitly for centuries in technical fields.

Categories And Objects …

First-order logic makes it easy to state facts about categories, either by relating objects to categories or by quantifying over their members. Here are some types of facts, with examples of each:

An object is a member of a category. $BB9 \in Basketballs$
A category is a subclass of another category.
$Basketballs \subset Balls$
All members of a category have some properties.
$(x \in Basketballs) \implies Spherical(x)$
Members of a category can be recognized by some properties. $Orange(x) \wedge Round(x) \wedge Diameter(x) = 9.5 \wedge xinBalls \implies x \in Basketballs$
A category as a whole has some properties.
$Dogs \in DomesticatedSpecies$

State relations between categories that are not subclasses

For example, if we just say that Males and Females are subclasses of Animals, then we have not said that a male cannot be a female. We say that two or more categories are disjoint if they have no members in common. And even if we know that males and females are disjoint, we will not know that an animal that is not a male must be a female, unless we say that males and females constitute an exhaustive decomposition of the animals. A disjoint exhaustive decomposition is known as a partition. The following examples illustrate these three concepts:

State relations between categories that are not subclasses …

```
Disjoint({Animals,Vegetables})
ExhaustiveDecomposition(
    {Americans,Canadians, Mexicans},NorthAmericans
)
Partition({Males, Females}, Animals)
```

State relations between categories that are not subclasses ...

Note that the `ExhaustiveDecomposition` of `NorthAmericans` is not a `Partition`.

$$\texttt{Disjoint(s)} \iff (\forall_{c1,c2} c1 \in s \land c2 \in s \land c1 \neq c2 \implies Intersection(c1, c2) = \{\})$$

$$\texttt{ExhaustiveDecomposition(s,c)} \iff (\forall_{i,i} \in c \iff \exists_c 2c2 \in s \land i \in c2))$$

Categories can also be defined by providing nessecary and sufficient conditions for membership.

Physical composition

The idea that one object can be part of another is a familiar one. Objects can be grouped into PartOf hierarchies, reminiscent of the Subset hierarchy:

```
PartOf (Bucharest , Romania)
PartOf (Romania, EasternEurope)
PartOf (EasternEurope, Europe)
PartOf (Europe, Earth)
```

The PartOf relation is transitive and reflexive; that is,

$\texttt{PartOf(x,y)} \land \texttt{PartOf(y,z)} \implies \texttt{PartOf(x,z)}.$

$\texttt{PartOf(x, x)}.$

Measurements

In both scientific and commonsense theories of the world, objects have height, mass, cost, and so on. The values that we assign for these properties are called measures. Ordinary quantitative measures are quite easy to represent. We imagine that the universe includes abstract "measure objects", such as the length that is the length of a 1.5 inch line segment: For instance we can say the length is 1.5 inches or 3.81 centimeters. Thus, the same length has different names in our language. We represent the length with a units function that takes a number as argument.

$\texttt{Length(L1)} = \texttt{Inches(1.5)} = \texttt{Centimeters(3.81)}$

Conversion between units is done by equating multiples of one unit to another: $\texttt{Centimeters(2.54} \times \texttt{d)} = \texttt{Inches(d)}$

Similar axioms can be written for pounds and kilograms, seconds and days, and dollars and cents. Measures can be used to describe objects as follows:

```
Diameter (Basketball 12) = Inches(9.5)
ListPrice(Basketball 12) = \(19)d\inDays\impliesDurati
```

Note that \$(1) is not a dollar bill! One can have two dollar bills, but there is only one object named \$(1). Note also that, while Inches(0) and Centimeters(0) refer to the same zero length, they are not identical to other zero measures, such as Seconds(0).

Objects: Things and stuff

The real world can be seen as consisting of primitive objects (e.g., atomic particles) and composite objects built from them.

By reasoning at the level of large objects such as apples and cars, we can overcome the complexity involved in dealing with vast numbers of primitive objects individually.

There is, however, a significant portion of reality that seems to defy any obvious individuation—division into distinct objects.

We give this portion the generic name stuff. For example, suppose I have some butter and an aardvark in front of me. I can say there is one aardvark, but there is no obvious number of “butter-objects”, because any part of a butter-object is also a butter-object, at least until we get to very small parts indeed. This is the major distinction between stuff and things. If we cut an aardvark in half, we do not get two aardvarks.

With some caveats about very small parts that we will omit for now, any part of a butter-object is also a butter-object:

$b \in \mathit{Butter} \wedge \mathit{PartOf}(p, b) \implies p \in \mathit{Butter}$

We can now say that butter melts at around 30 degrees centigrade:

$b \in \mathit{Butter} \implies \mathit{MeltingPoint}(b, \mathit{Centigrade}(30))$

Events

Events

Situation calculus is limited in its applicability: it was designed to describe a world in which actions are discrete, instantaneous, and happen one at a time. Consider a continuous action, such as filling a bathtub. Situation calculus can say that the tub is empty before the action and full when the action is done, but it can't talk about what happens during the action. It also can't describe two actions happening at the same time—such as brushing one's teeth while waiting for the tub to fill. To handle such cases we introduce an alternative formalism known as event calculus, which is based on points of time rather than on situations.

Events ...

Event calculus reifies fluents and events. The fluent `At(Shankar,Berkeley)` is an object that refers to the fact of `Shankar` being in `Berkeley`, but does not by itself say anything about whether it is true. To assert that a fluent is actually true at some point in time we use the predicate `T`, as in `T(At(Shankar , Berkeley), t)`.

Events …

Events are described as instances of event categories. The event `E1` of Shankar flying from San Francisco to Washington, D.C. is described as:

$E1 \in$
$Flyings \wedge Flyer(E1, Shankar) \wedge Origin(E1, SF) \wedge Destination(E1, DC)$

If this is too verbose, we can define an alternative three-argument version of the category of flying events and say

$E1 \in Flyings(Shankar, SF, DC)$ We then use `Happens(E1, i)` to say that the event `E1` took place over the time interval `i`, and we say the same thing in functional form with `Extent(E1)=i`. We represent time intervals by a `(start, end)` pair of times; that is, `i = (t1, t2)` is the time interval that starts at `t1` and ends at `t2`.

Events ...

We then use `Happens(E1, i)` to say that the event `E1` took place over the time interval `i`, and we say the same thing in functional form with `Extent(E1)=i`. We represent time intervals by a `(start, end)` pair of times; that is, `i = (t1, t2)` is the time interval that starts at `t1` and ends at `t2`.

To complete set of predicates for one version of the event calculus is

```
T(f, t) Fluent f is true at time t
Happens(e, i) Event e happens over the time interva
Initiates(e, f, t) Event e causes fluent f to start
Terminates(e, f, t) Event e causes fluent f to ceas
Clipped(f, i) Fluent f ceases to be true at some po
Restored (f, i) Fluent f becomes true sometime durin
```

We assume a distinguished event, Start, that describes the initial state by saying which fluents are initiated or terminated at the start time. We define T by saying that a fluent holds at a point in time if the fluent was initiated by an event at some time in the past and was not made false (clipped) by an intervening event. A fluent does not hold if it was terminated by an event and not made true (restored) by another event.

Processes

The events we have seen so far are what we call discrete events - they have a definite structure. `Shankar`'s trip has a beginning, middle, and end. If interrupted halfway, the event would be something different - it would not be a trip from San Francisco to Washington, but instead a trip from San Francisco to somewhere over Kansas. On the other hand, the category of events denoted by Flyings has a different quality. If we take a small interval of `Shankar`'s flight, say, the third 20-minute segment (while he waits anxiously for a bag of peanuts), that event is still a member of Flyings. In fact, this is true for any subinterval.

Processes …

Categories of events with this property are called process categories or liquid event categories. Any process *e* that happens over an interval also happens over any subinterval: $(e \in Processes) \wedge Happens(e, (t1, t4)) \wedge (t1 < et2 < t3 < t4) \implies Happens(e, (t2, t3))$

Time intervals

Event calculus opens us up to the possibility of talking about time, and time intervals. We will consider two kinds of time intervals: moments and extended intervals. The distinction is that only moments have zero duration:

```
Partition({Moments, ExtendedIntervals}, Intervals )
i \in Moments \iff Duration(i) = Seconds(0)
```

The function Duration gives the difference between the end time and the start time.

$Interval(i) \implies Duration(i) = (Time(End(i)) - Time(Begin(i)))$

```
Time(Begin(AD1900)) = Seconds(0)
Time(Begin(AD2001)) = Seconds(3187324800)
Time(End(AD2001)) = Seconds(3218860800)
Duration(AD2001) = Seconds(31536000)
```

Fluents & Objects

Physical objects can be viewed as generalized events, in the sense that a physical object is a chunk of space–time. For example, USA can be thought of as an event that began in, say, 1776 as a union of 13 states and is still in progress today as a union of 50.

We can describe the changing properties of USA using state fluents, such as `Population(USA)`. A property of the USA that changes every four or eight years, barring mishaps, is its president. One might propose that `President(USA)` is a logical term that denotes a different object at different times.

Fluents & Objects …

Unfortunately, this is not possible, because a term denotes exactly one object in a given model structure. (The term `President(USA, t)` can denote different objects, depending on the value of `t`, but our ontology keeps time indices separate from fluents.) The only possibility is that `President(USA)` denotes a single object that consists of different people at different times.

Fluents & Objects …

It is the object that is George Washington from 1789 to 1797, John Adams from 1797 to 1801, and so on, as in Figure 12.3. To say that George Washington was president throughout 1790, we can write

$$T(Equals(President(USA), GeorgeWashington), AD1790)$$

We use the function symbol Equals rather than the standard logical predicate =, because we cannot have a predicate as an argument to T, and because the interpretation is not that GeorgeWashington and President(USA) are logically identical in 1790; logical identity is not something that can change over time. The identity is between the subevents of each object that are defined by the period 1790.

Fluents & Objects …

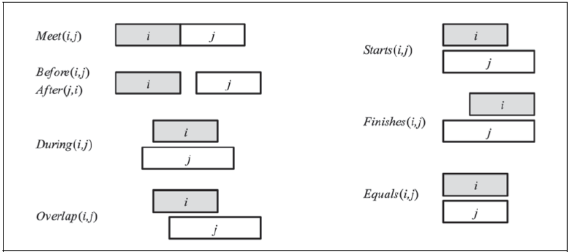

Figure 12.2 Predicates on time intervals.

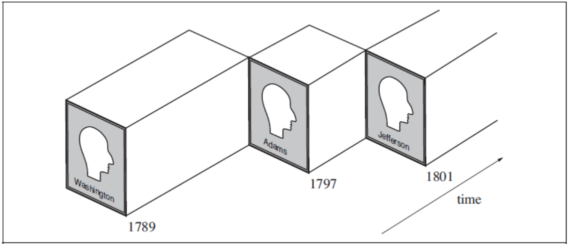

Figure 12.3 A schematic view of the object *President*(*USA*) for the first 15 years of its existence.

Mental Events and Mental Objects

Mental Events and Mental Objects

The agents we have constructed so far have beliefs and can deduce new beliefs. Yet none of them has any knowledge about beliefs or about deduction. Knowledge about one's own knowledge and reasoning processes is useful for controlling inference. For example, suppose Alice asks "what is the square root of 1764" and Bob replies "I don't know." If Alice insists "think harder", Bob should realize that with some more thought, this question can in fact be answered. On the other hand, if the question were "Is your mother sitting down right now?" then Bob should realize that thinking harder is unlikely to help. Knowledge about the knowledge of other agents is also important; Bob should realize that his mother knows whether she is sitting or not, and that asking her would be a way to find out.

We begin with the propositional attitudes that an agent can have toward mental objects: attitudes such as Believes, Knows, Wants, Intends, and Informs. The difficulty is that these attitudes do not behave like “normal” predicates. For example, suppose we try to assert that Lois knows that Superman can fly:

`Knows(Lois, CanFly(Superman)).`

One minor issue with this is that we normally think of `CanFly(Superman)` as a sentence, but here it appears as a term. That issue can be patched up just be reifying `CanFly(Superman);` making it a fluent.

A more serious problem is that, if it is true that Superman is Clark Kent, then we must conclude that Lois knows that Clark can fly:
$(Superman = Clark) \wedge Knows(Lois, CanFly(Superman)) \models Knows(Lois, CanFly(Clark))$