Today

Modelling time

Many beliefs depend on time – for example "it's raining" implicitly refers to a *present* time.

So, sometimes the same agent or person will accept the statement sometimes, reject it at others, and sometimes not be in a position to decide.

How can we represent such beliefs?

Belief statements

What is a suitable data structure to hold an agent's belief? Here is one way to do this.

For a given time, the agent may believe a non-temporal statement is *true*, *false*, or *unknown*. We assume that beliefs are *persistent*, i.e. in the absence of any other information, a statement believed to be true at some time is believed to be true at later times also.

Linear discrete unbounded time

The model we take for time here is that the time points are related by an order ("before") with the following properties:

- *linear*: the points succeed each other along a single line;
- *discrete*: for any point, there is a later point with no intermediate point between these two points;
- *unbounded*: there is no last point.

We can take the integers, or time given as a sequence of integers representing month/day/year/hours/minutes/seconds.

In both cases it's important to be able to compute the *later* relation, given two time points.

An abstract data-type

We cannot take rational numbers (not discrete), or integers with some maximum (bounded). We pass over implementational problems this raises.

We make a modular treatment of time if we do not specify the exact format, but only what operations we require. Then we can plug in different formats if required (e.g. include milliseconds).

Suppose we have:

```
add_time(Time1,Time2,Result)
less(Time1,Time2)
    % both assuming same format
inRange(Time1,Time2, Test)
    % is Test between 2 Times?
```

Changing beliefs

Use the following form to represent beliefs about `Pred`:

`b(A,Pred,[[Time1,t],[Time2,f],[Time3,t]])`

meaning that:

agent A started to believe Pred1 at Time1,
believes that Pred1 turns false at Time2,
and is true again for all times after Time3.
Before Time1, Agent1 has no belief about Pred1.

assumption: we always have that the `Times` in increasing temporal order; the implementation should maintain this invariant.

General Form

In general, label the statement with a list of pairs

$$[Time, Bod]$$

of time (in chosen format), and Boolean t/f.

We also ensure that the times mentioned are strictly increasing through the list. This means there is no contradiction, such as would arise with a label `[[114,t],[114,f]]` using the integer representation.

Note that we could equally have held a whole set of separate agent beliefs, one for each time point; this would have made computations more expensive.

Querying beliefs

Beliefs are added and queried with

`add_belief(Agent,[Time,Belief,TV]).`

`believes(Agent,[Time,Pred],TV).`

where `TV` can be given the value t or f (true or false), or return these values or "unknown". We do wnat to distinguish between a statement not being believed to be true (might be unknown), and the statement being believed to be false.

Let's see what we need to support these operations. In a Prolog implementation, we need to be concerned over the modes (instantiation patterns) that the operations are used in.

Time operations

We have some method of accessing the "current" time.

Recall we want to add times, compare them, and see if one is in a given range. The following modes are enough:

`add_time(+Time1,+Time2,-Result)`
`less(+Time1,+Time2)`
`inRange(+Time1,+Time2,+Test)`

For integers, these operations are straightforward. For month/day/year/hours/minutes/seconds, each in positive integers, indicate via a prioritisation, (here [2,3,1,4,5,6]), and do in terms of a hierarchy of levels.

Belief Operations

Given such time operations, how can we implement the updating of agents' beliefs? We think of this as changing the *state* of an agent. However, we also assume the agent remembers the former beliefs. In general, this issue is known as *belief revision* – how do we revise our opinions given new information?

Here we take a simple approach.

We use the Prolog assert and retract mechanism; this mimics the changing state of the agent in the internal Prolog database. To add a belief, there are two cases.

Belief addition

```
       % if no current belief, add as given
add_belief(Agent,[Time,Belief,TV]) :-
      not(b(Agent,Belief,_)),
      assertz(b(Agent,Belief,[[Time,TV]])).

       % if some belief already about fact,
       % work out new label for the fact.
       % and use that
add_belief(Agent,[Time,Belief,TV]) :-
      b(Agent,Belief,Old),
      add_to_belief_list(Old,Time,TV,New),
      retract(b(Agent,Belief,Old)),
      assertz(b(Agent,Belief,New)).
```


Updating Labels

We make the assumption that the only connective that can appear in a fact is negation. Thus, if we add a belief that $\neg raining$ is true at time 1137, we want to associate false with $raining$ at that time. We do allow function symbols and constants.

On the other hand, if we want to query for $raining$ at 1136, and there is no other information, we want to getback the "don't know" answer. We want the same answer if we query for $\neg raining$ at 1136 too.

We may also want to query if an agent does *not* believe some (possibly negated) fact!

Updating Labels (ctd)

Suppose that we have reduced the problem so that we have an appropriate boolean label, and the existing label has times in order. Suppose first time in current label is $T1$ and new time is TN.

- If TN is less than $T1$, add new pair to front;
- If TN is same as $T1$, replace first pair with new pair;
- Otherwise keep first pair as first, and call recursively.

The base case is the empty list of pairs — simply insert the new pair to make a list with one pair.

Querying beliefs

It is helpful to work out first the case where the belief being queried has no negation.

Suppose we have the label list for a statement, and we want to see what of the three truth values we associate with a time t,

Implement `no_not_believes` via cases here:

- when t is given
  – when list is empty
  – when t is not less than first label time
  – otherwise
- when time t is a (Prolog) variable (i.e. when we want to compute a value for the unkown t).

Including Negations

First, we need to consider the case where there is no belief at all about the statement queried.

Finally, we query beliefs by looking at unnegated form, and then dealing with negation subsequently:

```
believes(Agent,[Time,Belief],NTV) :-
     remove_nots(Belief,NewBel),
     no_nots_believes(Agent,[Time,NewBel],TV),
     add_in_nots(Belief,TV,NTV).
```

– beware that we have the "unknown" truth value to deal with here.

Three valued logic

The reasoning above with three truth values (true, false, unknown) has been extensively studied. We can think of this as using extended truth tables; here for negation we have:

$\neg$	
t	f
f	t
u	u

Can also do for other connective, e.g. conjunction:

$\wedge$	t	f	u
t	t	f	u
f	f	f	f
u	u	f	u

Prolog and negation

In implementing such queries, we have introduced explicit truth values into the language we manipulate, rather than relying on Prolog's own notion of success and failure.

This is because Prolog does not distinguish between being unable to find a derivation, and claiming that the query is false; that is, it does not distinguish between the "false" and the "unknown" values we have above.

When we take a Prolog reponse of `no.` as indicating that a query is false, we are making use of the idea of *negation as failure*: if a statement cannot be derived, then it is false.

Clearly, this assumption is not always valid!

Example

In fact, in the declarative reading, a Prolog query will fail if there is no deduction of the query from the program, read as a set of logical assertions in Horn clause shape.

If some information is not present in the program, failure to find a derivation should not let us conclude that the query is false – we just don't have the information to decide.

Summary

- Linear discrete time
- Representing changing beliefs
- Adding and querying beliefs
- Prolog and negation as failure