University of Malta
BSc(Hons.) IT Year II

Problem Solving Techniques in Artificial Intelligence

Lecture 12.1

Truth Maintenance

- Reason Maintenance Systems

Department of Computer Science and AI
1998/9

Reason Maintenance

When a forward chaining rule triggers, no record is kept of the triggering conditions.

The effects of the rule are not undone if the triggering conditions subsequently change.

Sometimes this is appropriate:

If win(X,P) and value(P,100000) then lucky(P)

but sometimes not

If see(R) and robot(R) then currently_visible(R)

since in the latter case, as soon as R goes out of sight he is no longer visible.

Clearly, in this latter case different behaviour is required. The role of reason maintenance systems (RMSs) is to provide the extra functionality.

Justifications and Premises

RMS if-then rules are called *justifications*. We can represent justifications just as we did forward-chaining rules, except for renaming the operators.

op(xfx,1150, —:).

We restrict the discussion here to *ground justifications*, that is, justifications that contain no variables. In addition, we assume that all RHSs have positive polarity. The general form of justification rules will therefore be:

(d1,s1),...,(dn,sn) —: d

where the LHS items (or *justifiers*) are database items, each si is a polarity and the RHS conclusion is the *justificand*. An example of such a justification rule would be

(see(robot),+),
(friendly(robot),-) —: threat(robot).

Monotonicity

If all the conditions on the left are positive the justification is called *monotonic*. Otherwise it is *non-monotonic*.

The reason for this is that if all the conditions are positive, the rhs is monotonic in the database in the sense that it cannot be removed as a result of *adding* something to the database.

Standard logic is monotonic in the sense that the set of theorems derivable within a given system can only grow as more axioms are added.

Yet this is not always a very good model of human reasoning, which often employs *default knowledge* of the form "believe P unless you gain information to the contrary".

An Example of Non-Monotonic Reasoning

All birds can fly.
Tweety is a bird

Therefore, Tweety can fly.

All birds can fly.
Tweety is a bird
Tweety is a penguin

Therefore, tweety cannot fly.

Addition of the fact that Tweety is a penguin invalidates the original conclusion

RMS versus Production System

A forward chaining production system may entertain beliefs called facts that are not supported by any rule

RMSs, on the other hand require that every belief be *supported* by some explicitly stated justification rule.

All beliefs must be grounded in a justification whose lhs is itself justified.

This recursive definition must bottom out in some beliefs that are held unconditionally. These beliefs are called *premises*.

Support by Justifications

In the example to follow, there are three beliefs under consideration. Each has a *status* indicated by a *label* IN or OUT.

```
          IN: threat(robot)
                 ^
                 |
       ---------X---------
      |                  |
      |+                 |-
      |                  |
IN: see(robot)      OUT: friendly(robot)
      ^
      |
      X
      |+
   premise
```

IN means that the item is “in” the database. OUT means the reverse. N.B. IN and OUT are logically distinct from true and false though they can be used to model true and false beliefs.

Well Founded Support

Each belief is supported by (i.e. is the justificand of) a justification. The two justifications above are marked X.

Nodes which are unsupported are OUT.

Support for a belief is said to be *well-founded* if every justifier has a polarity that is supported by the the status of the underlying database item, i.e.

- A justifier marked $+$ is suported by a belief that is IN
- A justifier marked - is supported by a belief that is OUT

Exercise. Convince yourself that the belief "threat(robot)" has well-founded support in the above network.

Operations on RMSs

- In the case of forward chaining we considered the operations of adding and deleting *items*, whilst the set of forward chaining rules was considered to be fixed.

- In the case of RMSs, the situation is reversed: here, the set of dabatase items is fixed, and the *only* operations we consider are the addition and deletion of *justifications*.

- As a special case we may add or delete an item whose justifier is the special item **premise**.

- These additions and deletions may then have ramification throughout the network until all and only the justified nodes are actually in the database.

Example 1

In example 1 all and only the justified nodes are labelled IN.

```
IN: has_feather                IN: can_fly
     |                              |
     X                              X---
     |                              |   |
     ----------------  -------------    |
                   +|  |+               |
                    |  |                |
                IN: is_bird             |
                    |  |                |
                    |  |                |
                    X  X------------    |
                    |               |   |
                    |               |   |
                    |+             +|   |-
                 premise       OUT: is_penguin
```

Example 2

Starting with the situation just depicted, Example 2 shows the result of adding a justification that makes is_penguin become IN.

```
IN: has_feather                OUT: can_fly
     |                              |
     X                              X---
     |                              |   |
     ----------------  -------------    |
                   +|  |+               |
                    |  |                |
                IN: is_bird             |
                    |  |                |
                    |  |                |
                    X  X------------    |
                    |               |   |
                    |               |   |
                    |+   +         +|   |-
                 premise -----X----IN:is_penguin
```

Example 3

Example 3 shows the result (again starting with example 1) of deleting the justification for is_bird.

```
OUT: has_feather                OUT: can_fly
     |                              |
     X                              X---
     |                              |   |
     ----------------  -------------    |
                   +|  |+               |
                    |  |                |
                OUT: is_bird            |
                       |                |
                       |                |
                       X------------    |
                                    |   |
                                    |   |
                                   +|   |-
                               OUT: is_penguin
```

Non Monotonicity

- We should note that adding a justification can result in the removal of beliefs, since some beliefs rely on the absence of other beliefs.
- In other words, the set of database items is not a monotonically increasing function of the set of justifications.

Adding or Removing Justifications

If the justification network contains no cycles, the following procedure for adding (deleting) a justification will suffice:

1. Add (delete) the appropriate justification from the database.
2. Determine whether the justificand should be in the database on the basis of the new (remaining) justifications.
3. If it should (should not), add the item to the database (delete item from database), treating all justifications as forward chaining rules.

Cycles

In the presence of cycles the situation is not so straightforward. In particular, three types of anomalous situation can arise:

- Self Justification
- Underdeterminacy
- Overdeterminacy

Self Justification

Consider the a network with the following initial configuration:

```
 ----->IN:Q------
|            +    |
|                 |
X                 X
|                 |
|      +          |
 ------IN:P <----
         |
         X
         |
         |+
     premise
```

Now suppose the justification premise —: p is removed.

Self Justification

In the resulting network

```
 ----->IN:Q------
|           +    |
|                |
X                X
|                |
|     +          |
 ------IN:P <----
```

P and Q justify each other, but neither is grounded.

Underdeterminacy

Another problem with cycles is underdeterminacy. In the following network P and Q must have opposite polarity, but there are two possible solutions.

```
 ----->??:Q------
|           -    |
|                |
X                X
|                |
|     -          |
 ------??:P <----
```

Overdeterminacy

Finally, it is not hard to see that the following network is overdetermined: there is nomconsistent labelling.

```
 ----->??:Q------
|            -    |
|                 |
X                 X
|                 |
|     +           |
 ------??:P <----
```