University of Malta
BSc(Hons.) IT Year II

Problem Solving Techniques in Artificial Intelligence

Lecture 12.2

Truth Maintenance

- Reason Maintenance (Implementation)

Department of Computer Science and AI
1998/9

Terminology

```
JUSTIFICATION:

justifier content
 |
 |  justifier polarity
 |  |
 |  |
 |  |
(P1,S1),....,(Pn,Sn)  ----: P
--------------------       ---
        |                   |
        |                   |
    justifiers (LHS)        |
                            |
                       justificand
```

Implementing Justifications

Justifications are very similar to forward chaining rules. We adopt a similar representation scheme whereby the justification is broken down into several "if" facts and one "then" fact. We use the predicates justifier/3 and justificand/3 to do this. For example, the justification

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

might would be represented

```
justifier(see(robot),+,j12).
justifier(friendly(robot),-,j12).
justificand(j12,1,threat(robot)).
```

where j12 is a unique identifier.

Adding/Removing a Justification

- Whenever we add or remove a justification, the IN/OUT status of the its justificand is put into question, since it might change. We will say that the justificand node (i.e. has become *contaminated*.
- This node may in turn contaminate other nodes, since the justificand of a justification containing contaminated nodes itself becomes contaminated.
- The procedure therefore first determines the set of contaminated nodes.
- It then considers which contaminated nodes can be *revalidated* in virtue of justifications whose LHS contains no contaminated nodes.

The Algorithm: Add Justification

```
% If justification already present, do nothing
add_just_rms_slow(If,Then) :-
   exist_just_rms_slow(If,Then,_), !.

add_just_rms_slow(If,Then) :-
   new_just_label(J),
   findall( (Justifier,S),
            (amember((Justifier,S),If),
             assert(justifier(Justifier,S,J)),
             unsatisfied(Justifier,S)),
            UnSat),
   length(UnSat,N),
   assert(justificand(J,N,Then)),
   mark_contaminated_nodes_slow(Then),
   revalidate_rms_slow.


satisfied(Item,+)   :- clause(rms(Item),true).
satisfied(Item,-)   :- not(satisfied(Item,+)).
unsatisfied(Item,S) :- not(satisfied(Item,S)).
```

Delete Justification

```
del_just_rms_slow(If,Then) :-
        exist_just_rms_slow(If,Then,J), !,
        retractall(justifier(_,_,J)),
        retract(justificand(J,_,_)),
        mark_contaminated_nodes_slow(Then),
        revalidate_rms_slow.
del_just_rms_slow(_,_).
```

Finding Contaminated Nodes

```
% To mark the contaminated nodes,
% perform depth-first search,
% using justifications as arcs between nodes:
mark_contaminated_nodes_slow(Node) :-
        retractall(contaminated(_)),
        retractall(contam_marked(_)),
        do_all(( contam_path(Node,Node1),
                 assert(contaminated(Node1)) )).

contam_path(Node,Node).
contam_path(Node,Node1) :-
        justifier(Node,_,J),
        justificand(J,_,NextNode),
        not(contam_marked(NextNode)),
        assert(contam_marked(NextNode)),
        contam_path(NextNode,Node1).
```

Revalidating Nodes

For every contaminated node:

- If is has a determined polarity, update its status.
- If is not determined, assume negative polarity, and update its status.

```
revalidate_rms_slow :-
        contaminated(Node),
        determined(Node,S), !,
        retract(contaminated(Node)),
        update_node_rms_slow(Node,S),
        revalidate_rms_slow.
revalidate_rms_slow :-
        retract(contaminated(Node)), !,
        update_node_rms_slow(Node,-),
        revalidate_rms_slow.
revalidate_rms_slow.
```

Determined Nodes

A node is determined if

- Its polarity is positive and it is supported by a justification whose justifier counter is zero.
- Its polarity is negative and every justification for it has some uncontaminated unsatisfied justifier.

```
determined(Node,+) :-
        justificand(J,0,Node),
        not(( justifier(Justifier,_,J),
              contaminated(Justifier)) ).
determined(Node,-) :-
        not(( justificand(J,_,Node),
              not(( justifier(Justifier,S,J),
                    unsatisfied(Justifier,S),
                    not(contaminated(Justifier)) )) )).
```

Update Node

```
update_node_rms_slow(Node,S) :-
  satisfied(Node,S), !.
update_node_rms_slow(Node,S) :-
  adjust_status(Node,S),
  do_all(( justifier(Node,S1,J),
           once(retract(justificand(J,N,NextNode))),
           update_counter(S,S1,N,M),
           assert(justificand(J,M,NextNode)) )).


update_counter(S,S,N,M) :- M is N-1, !.
update_counter(_,_,N,M) :- M is N+1.

satisfied(Item,+)   :- clause(rms(Item),true).
satisfied(Item,-)   :- not(satisfied(Item,+)).

adjust_status(Node,+) :- assert(rms(Node)).
adjust_status(Node,-) :- retract(rms(Node)).
```

Results

```
% | ?- test.
% *****  START ******
% Database listing :
% The facts:
% rms(premiss).
%
% Justifiers:
% Justificands:
%
% Add justification  "(bird, +)---: fly" ******
% Database listing :
% The facts:
% rms(premiss).
%
% Justifiers:
% justifier(bird,+,j1).
%
% Justificands:
% justificand(j1,1,fly).
%
```

Add Justification 2

```
% Add justification
% " (has_beak,+),(has_wing,+)---: bird"
% Database listing :
% The facts:
% rms(premiss).
%
% Justifiers:
% justifier(bird,+,j1).
% justifier(has_beak,+,j2).
% justifier(has_wing,+,j2).
%
% Justificands:
% justificand(j1,1,fly).
% justificand(j2,2,bird).
```

Add Premises

```
% Add justification  "(premiss,+) ---: has_beak "
% Add justification  "(premiss,+) ---:  has_wing"
% Database listing :
% The facts:
% rms(premiss).
% rms(has_beak).
% rms(has_wing).
% rms(bird).
% rms(fly).
%
% Justifiers:
% justifier(bird,+,j1).
% justifier(has_beak,+,j2).
% justifier(has_wing,+,j2).
% justifier(premiss,+,j3).
% justifier(premiss,+,j4).
%
% Justificands:
% justificand(j3,0,has_beak).
% justificand(j4,0,has_wing).
% justificand(j2,0,bird).
% justificand(j1,0,fly).
```

Retract Premise

```
% Delete justification
% "(premiss,+) ---:  has_beak "
% Database listing :
% The facts:
% rms(premiss).
% rms(has_wing).
%
% Justifiers:
% justifier(bird,+,j1).
% justifier(has_beak,+,j2).
% justifier(has_wing,+,j2).
% justifier(premiss,+,j4).
%
% Justificands:
% justificand(j4,0,has_wing).
% justificand(j2,1,bird).
% justificand(j1,1,fly).
```

Add Premises

```
% Add just.  " (premiss,+) ---: has_beak " ******
% Add just.  " (premiss,+) ---:  has_wing" ******
% Database listing :
% The facts:
% rms(premiss).
% rms(has_wing).
% rms(has_beak).
% rms(bird).
% rms(fly).
%
% Justifiers:
% justifier(bird,+,j1).
% justifier(has_beak,+,j2).
% justifier(has_wing,+,j2).
% justifier(premiss,+,j4).
% justifier(premiss,+,j5).
%
% Justificands:
% justificand(j4,0,has_wing).
% justificand(j5,0,has_beak).
% justificand(j2,0,bird).
% justificand(j1,0,fly).
```

Delete Justification

```
% Delete justification
% " (has_beak,+),(has_wing,+)---: bird" ******
% Database listing :
% The facts:
% rms(premiss).
% rms(has_wing).
% rms(has_beak).
%
% Justifiers:
% justifier(bird,+,j1).
% justifier(premiss,+,j4).
% justifier(premiss,+,j5).
%
% Justificands:
%justificand(j4,0,has_wing).
%justificand(j5,0,has_beak).
%justificand(j1,1,fly).
```

Consistency Maintenance

So far the only constraint on the TMS database was that all and only the justified items be in it.

We now introduce a further constraint: that the database be consistent.

A priori, there are no constraints on what is considered to be inconsistent.

- We may consider friendly(robot) and threat(robot) as inconsistent
- We might permit any pair from the set {wine, women, song} to be consistent but not all three together.
- As a special case of a binary inconsistency, we might consider that for any item I, there is another item -I (i.e. the "negation" of the item) which is inconsistent with it.

Consistency Maintenance Systems

These sets of forbidden items are called *no-good sets*.

The task of a consistency maintenance system (CMS) is to "look over the shoulder" of the RMS and ensure that all nogoods are respected: to make sure that there does not exist a nogood all of whose members are in the database.