University of Malta
BSc(Hons.) IT Year II

Problem Solving Techniques in Artificial Intelligence

Lecture 10.2

Inference Engines

- Forward Chaining
- Production Systems

Department of Computer Science and AI
1998/9

Forward Chaining

Consider an if-then rule such as

R1: IF you believe both p and q THEN believe r

One way to use this rule is to wait until a query about r is asked, and replace it with a query about p and q.

This is a goal driven, backward chaining use of the rule as seen in Prolog and other backward chaining systems.

There is, however, another way to use the rule, namely, as soon as p and q are known to be true in the knowledge base, then add r.

This is a *data-driven, forward-chaining* use of the rules which may have *ramifications* if there are other rules present in the database such as

R2: If you believe both r then believe s

Representing Forward Chaining Rules in Prolog

We'll start by defining the operator **=>**

```
op(1100,xfx,=>)
```

For the moment we will restrict ourselves to rules which contain no variables and which only mention facts present in the database on the lhs and facts to be added on the rhs. Later we will relax these assumptions.

Examples of such rules are

```
r   => q.
r,s => p.
q   => t.
```

Forward Chaining as a Special Case

As described so far, forward chaining involves addition of a fact as a result of addition of another fact. That is: an addition event triggering an addition event

We should note in passing that this is something of a special case. For instance, we could also envisage the following possibilities.

- A deletion event triggering an addition event
- A deletion event triggering a deletion event
- An addition event triggering a deletion event.

We can also envisage recording the depedency of the conclusion upon the premise, so that if the premise is later removed, so is then conclusion. This is the basis of *reason maintenance systems*

Addition of Database Items

Suppose we wish to model

1. Addition of a new idem to the database.
2. Propagation of its effects

Assuming that rules have at most a single condition on the lhs, one definition would be as follows:

if the fact is already there, do nothing.

```
assert_forward0(X) :- clause(X,true), !.
```

otherwise assert it, find all triggered rules, and propagate

```
assert_forward0(X) :- asserta(X),
                      (X =>B),
                      assert_forward0(B),
                      fail.
assert_forward0(_).
```

Complete Program

```
:- op(1150,xfx,=>).

p   => q.
r,s => p.
q   => t.

assert_forward0(X)  :- clause(X,true), !.
assert_forward0(X)  :- asserta(X),
                  (X =>B),
                  assert_forward0(B),
                  fail.
assert_forward0(_).

PROGRAM TRACE
2 ?- t.
[WARNING: Undefined predicate: `t/0']
No
3 ?- assert_forward0(p).
Yes
4 ?- t.
Yes
```

Rules with Several L.H. Conditions

```
R2. r,s => p.
```

`assert_forward0` is unable to handle rules such as R2 that contain more than one condition. The following program addresses the problem:

```
assert_forward1(X) :- clause(X,true), !.
```

find all rules whose lhs contains X, make sure all other LHS items are in the database, and apply **assert_forward1** *their rh sides*

```
assert_forward1(X) :-
  asserta(X),
  (If => Then),
  amember(X,If),
  not((amember(Y,If),
       not(clause(Y,true)))),
  assert_forward1(Then),
  fail.

assert_forward1(_).
```

`amember(X,CL)`

`amember/2` is just like **`member/2`** except that whilst **`member`** works with an element and a Prolog list, **`amember`** works with a "comma list".

```
amember(X,(A,B)) :-
 !,
 (X=A ; amember(X,B)).
amember(X,X).
```

```
?- amember(X,(a,b,c)).
X = a ;
X = b ;
X = c ;
```

Production Systems

- A production system consists of a *working memory* and a *production memory*.
- Working memory is a collection of items, an item being any *datum*.
- Roughly, a datum corresponds to a Prolog term that is *ground* i.e. does not contain any variables.
- Production memory is a set of *production rules*. These are generalised If-Then rules with a LHS and and RHS.

Production Systems

- In the simplest systems the If part is a collection of conditions and the Then part is a simple assertion.
- In more complex systems the Then part might consist of several assertions, and arbitrarily complex *programs* can be associated with both LHS conditions and RHS items.
- Almost all such programs have to do with adding, deleting, or modifying the contents of working memory, and sometimes with addition, deletion and modification of production memory itself.

Operation of Production Systems

Given these two data structures, the operation of the system is defined by the following cycle:

1. Determine the *conflict set*: Find all rules whose LHS is satisfied.
2. Resolve the conflict: select one of the rules from the conflict set, and discard the rest.
3. Act, or "fire": execute the action part of the selected rule.
4. Go to 1.

Remarks

The operation of a production system is therefore similar to that of a forward chainer. Both bring about changes to the database on the basis of data already present and If-Then rules.

The most important difference that whereas a forward chaining blindly fires all the rules that triggered and then quits, a production system carefully selects one rule to fire, ignores the rest, and then repeats the process.

A less essential difference is that a production system usually specifies a *goal*. As soon as the goal appears in the working memory, the system stops.

Top Level Architecture

```
ps :-
 conflictset(CS),
 resolve_set(CS,ID/RHS),
 eval_rhs(RHS),
 ps.

ps.
```

Checkpoint

- Why are there two clauses for ps?
- How would you modify ps to accept a goal.

Discovering the Conflict Set

```
conflictset(CS) :-
 findall(X/RHS,satisfied(X,RHS),CS).

satisfied(RuleID,RHS) :-
 (RuleID :: LHS => RHS),
 eval_lhs(LHS,RHS),
 not(_ :: => RHS).

eval_lhs((X,Y),RHS) :-
    !,
    ( _ :: => X),
    eval_lhs(Y,RHS).
eval_lhs(LastCond,RHS) :-
    ( _ :: => LastCond).
```

Operation

```
?- ps.
adding fact _G245:: =>father(daniel, mike)
adding fact _G291:: =>father(mike, maurice)
adding fact _G337:: =>grandfather(daniel, maurice)

Yes
```

Generalised Production Systems

- Explicit goals
- Deletion of database items.
- Recording justifications.