IKI30820 – Logic Programming
Negation as Failure
Slide 06

Ari Saptawijaya (slides) Adila A. Krisnadhi (LaTeXadaptation)

Fakultas Ilmu Komputer
Universitas Indonesia

2009/2010 • Semester Gasal

Outline

Motivation

- One of Prolog's most useful features is the simple way it lets us to state generalizations.
  - Dhani enjoys movies:
    ```
    enjoys(dhani,X) :- movie(X).
    ```
- But, in real life rules have exception.
  - Dhani enjoys movies, except horror movies.
- How do we state this in Prolog?

`fail`

- That something is not true can be said in Prolog using a special goal: **`fail`**
  - **`fail`** is a goal that always fails
  - It can be used to force backtracking
- In combination with 'cut' (that blocks backtracking), 'fail' enables to state the previous situation (i.e. Dhani's preferences of movies)

Example

```
enjoys(dhani,X) :- horror(X), !, fail.
enjoys(dhani,X) :- movie(X).
movie(X) :- action(X).
movie(X) :- drama(X).
movie(X) :- horror(X).
action('Die Hard').          drama('Amelie').
horror('The Ring').

?- enjoys(dhani,'Die Hard').
Yes

?- enjoys(dhani,'Amelie').
Yes

?- enjoys(dhani,'The Ring').
No
```

not

- It is useful to have a unary predicate 'not'
  - `not(Goal)` is true if `Goal` is not true.
  - Def.: If `Goal` succeeds then `not(Goal)` fails, otherwise `not(Goal)` succeeds.

```
not(Goal) :- Goal,!,fail.
not(Goal) :- true.
```

- 'true' is a goal that always succeeds.
- In SWI-Prolog, 'not' can be written as `\+`

- Dhani's preference on movies (the first two clauses) can now be written as follows, i.e. using 'not' or `\+`

```
enjoys(dhani,X) :- movie(X), \+horror(X).

?- enjoys(dhani,X).
X = 'Die Hard' ;
X = 'Amelie' ;
No
```

Be careful ...

- As also cut, 'not' should also be used with care.
- 'not' does **not** correspond exactly to logical negation.
  Suppose we have:

```
enjoys(dhani,X) :- \+horror(X), movie(X).
```

  instead of:

```
enjoys(dhani,X) :- movie(X), \+horror(X).
```

  What is the answer of the query:

```
?- enjoys(dhani,X).
```

- Now, the answer of the query:

```
?- enjoys(dhani,X).
No
```

  - the first thing that Prolog has to check is whether `\+horror(X)` holds, which means that it must check whether `horror(X)` fails.
  - But this succeeds! The database contains the information: horror(The Ring').
  - So the query `\+ horror(X)` fails, hence the original query does too.

- The crucial difference:
  - the original version (the 2nd version — the one that works right): `\+` is used only **after** we have instantiated the variable `X`.
  - the new version (the 1st version — which goes wrong): `\+` is used **before** we instantiate the variable X

Closed world assumption

- The reasoning of 'not' is based on **closed world assumption**.
- The world is closed: everything that exists is stated in the program or can be derived from the program.
- If something is not in the program (or cannot be derived from it), then it is not true. Consequently: the negation is true.

```
hobbit(frodo).
elf(legolas).

?- hobbit(gollum).
No.
?- \+ hobbit(gollum).
Yes.
```

- In the usual interpretation:

  `?- hobbit(X)`

  means: "Does there ***exist*** `X`, such that `hobbit(X)` is true? If yes, what is `X`?"
  - So, X is existentially quantified.
  - Prolog's answer: `X = frodo.`
- On the other hand,

  `?- \+ hobbit(X)`

  is **not** interpreted as: "Does there exist `X`, such that `hobbit(X)` is not true?"
  - Expected answer: X = legolas.
  - Prolog's answer: No
  - Prolog's interpretation: "It is not true, that there exists `X`, s.t. `hobbit(X)`", which is equivalent to: "for all `X`: not `hobbit(X)`".

Outline

```
attend(budi,lp).
attend(denvil,lp).
attend(erwin,lp).
attend(iqbal,lp).
attend(budi,citra).
attend(denvil,citra).
attend(joji,citra).
attend(afif,citra).
```

Who attends `citra`, but not `lp`?

```
?- attend(X,citra), \+attend(X,lp).
```

```
zero(0).
positive(X) :- \+zero(X).

?- positive(0).

?- positive(1).

?- positive(X).
```

```
zero(0).
positive(X) :- num(X), \+zero(X).
num(0).
num(s(X)) :- num(X).

?- positive(0).

?- positive(s(0)).

?- positive(X).
```

Example: Eight Queens Problem

```
% solution(BoardPosition) if BoardPosition is a list
% of non-attacking queens

solution([]).
solution([X/Y|Others]) :-     %1st queen at X/Y,
                              %other queens at Others
  solution(Others),
  member(Y,[1,2,3,4,5,6,7,8]),
  \+ attacks(X/Y, Others).      %1st queen does not
                                %attack others
attacks(X/Y,Others) :-
  member(X1/Y1,Others),
  (Y1 =:= Y;                    %same Y-coordinates
   Y1-Y =:= X1-X;               %same diagonal
   Y1-Y =:= X-X1).

% a solution template
template([1/Y1,2/Y2,3/Y3,4/Y4,5/Y5,6/Y6,7/Y7,8/Y8]).
```

```
?- template(X), solution(X).
X = [1/4, 2/2, 3/7, 4/3, 5/6, 6/8, 7/5, 8/1] ;
X = [1/5, 2/2, 3/4, 4/7, 5/3, 6/8, 7/6, 8/1] ;
X = [1/3, 2/5, 3/2, 4/8, 5/6, 6/4, 7/7, 8/1] ;
...
Yes
```