Computational Logic

Introduction to Logic Programming

Damiano Zanardini

UPM European Master in Computational Logic (EMCL)
School of Computer Science
Technical University of Madrid
damiano@fi.upm.es

Academic Year 2009/2010

Programs

Terminology

- a Horn clause without negated literals is a *fact*
- a Horn clause which has a non-negated literal and the other literals negated is a *rule*
  - the non-negated literal is the *head*
  - the sequence of negated literals is the *body*
- a set of rules with the same head predicate p is a *procedure*, whose name is p
- a *logic program* is a set of procedures

Rule syntax

$A \vee \neg B_1 \vee \neg B_2$	$B_1 \wedge B_2 \rightarrow A$	$A \leftarrow B_1, B_2.$	`A :- B1,B2.`
A	A	$A.$	`A.`

Queries

Execution

- a *goal* is a Horn clause with all literals negated
- the deduction whose correctness has to be verified has the program as premise, and the goal as conclusion
- the *execution* of a logic program on a given goal consists of verifying if the goal can be deduced from the program, and, if it can, computing the values of the goal variables which give the answer
- SLD resolution is used, with the goal as the initial goal clause
- most implementations of this kind of languages use a computation rule which follows the order in which rules are written (top-down) and depth search with *backtracking*
  - infinite loops may occur

Query syntax

$\neg B_1 \vee \neg B_2$	$B_1 \wedge B_2$	$\leftarrow B_1, B_2.$	`?- B1,B2.`

Properties

Limitations: some rules are not allowed

- $P_1 \wedge P_2 \rightarrow \neg Q$
  - implication cannot end in something which is not true
- $P_1 \wedge P_2 \rightarrow Q_1 \vee Q_2$
  - it is not possible to state a disjunction
- $P_1 \wedge \neg P_2 \rightarrow Q$
  - premises must be true

Negation

- complete knowledge about the universe is assumed (*closed-world hypothesis*)
- negation is simulated by *negation as failure*: what cannot be proven is false
  - dangerous, but useful in finite domains

Execution

To *prove* a literal C

1. put C and the corresponding answer literal in a stack $\mathcal{S}$
2. repeat until the top of $\mathcal{S}$ is an answer literal and no further steps can be preformed
    1. pop from $\mathcal{S}$ a literal L
    2. choose a rule or fact whose head unifies with L (MGU α)
        - push in $\mathcal{S}$ the literals (ordered) of the body of the rule
        - apply α to the complete $\mathcal{S}$
        - rename variables in $\mathcal{S}$
    3. if not possible, *fail*

Backtracking

When the choice of the rule whose head unifies with L comes to be impossible, the search must go *back* to a *choice point* above in the tree and take another literal L'

Example

Parents and grandparents

```
1   father(a,b).
2   mother(b,c).
3   grandparent(X,Z)  :-  parent(X,Y), parent(Y,Z).
4   parent(X,Y)       :-  father(X,Y).
5   parent(X,Y)       :-  mother(X,Y).
```

- who is the grandparent of *c*? `?- grandparent(X,c).`

`grandparent(X,c), ans(X).`	$\rightsquigarrow$	$3, \{X_3/X, Z_3/C\}$
`parent(X,Y3), parent(Y3,c), ans(X)`	$\rightsquigarrow$	$4, \{X_4/X, Y_4/Y_3\}$
`father(X,Y3), parent(Y3,c), ans(X)`	$\rightsquigarrow$	$1, \{X/a, Y_3/b\}$
`parent(b,c), ans(a)`	$\rightsquigarrow$	$4, \{X_4'/b, Y_4'/c\}$
`father(b,c), ans(a)`	$\rightsquigarrow$	$\mathit{fail}, 5, \{X_5/b, Y_5/c\}$
`mother(b,c), ans(a)`	$\rightsquigarrow$	$2, \{\}$
`ans(a)`		

Example

Parents and grandparents

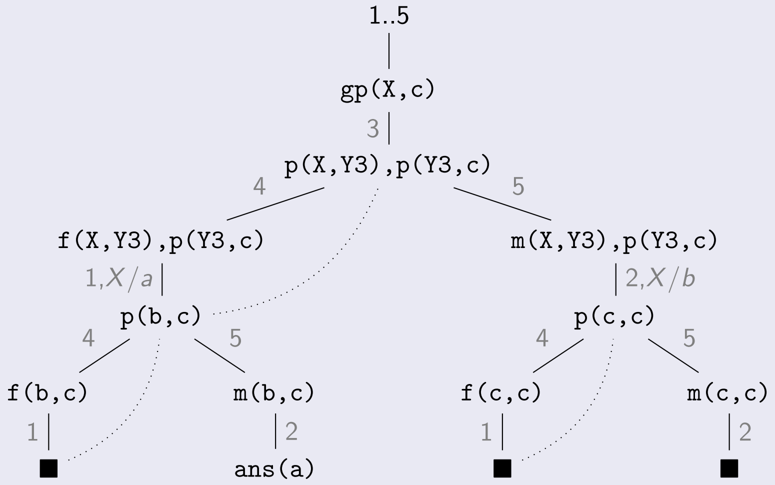
```
1  father(a,b).
2  mother(b,c).
3  grandparent(X,Z)  :-  parent(X,Y), parent(Y,Z).
4  parent(X,Y)       :-  father(X,Y).
5  parent(X,Y)       :-  mother(X,Y).
```

- who is the grandchild of *a*? `?- grandparent(a,X).`

`grandparent(a,X), ans(X).`	$\rightsquigarrow$	$3, \{X_3/a, Z_3/X\}$
`parent(a,Y3), parent(Y3,X), ans(X)`	$\rightsquigarrow$	$4, \{X_4/a, Y_4/Y_3\}$
`father(a,Y3), parent(Y3,X), ans(X)`	$\rightsquigarrow$	$1, \{Y_3/b\}$
`parent(b,X), ans(X)`	$\rightsquigarrow$	$4, \{X_4'/b, Y_4'/X\}$
`father(b,X), ans(X)`	$\rightsquigarrow$	$\mathit{fail}, 5, \{X_5/b, Y_5/X\}$
`mother(b,X), ans(X)`	$\rightsquigarrow$	$2, \{X/c\}$
`ans(c)`		

Example

Parents and grandparents



Operational vs. declarative

Operational

- the program defines a series of *procedures* (the heads) using *calls* to other procedures (the literals in the body)
- the goal is a series of calls to be executed sequentially (in the order they are written), with the possibility to *go back*

Declarative

- the program declares the information about the problem to be solved
- the problem is formulated as a question
- the task is proving that the question is a correct conclusion of the premises (program)
- an execution is a proof

Operational vs. declarative

Applications

- arithmetics (reversible)
- data structures, recursion
- database systems
- search problems
- rule-based expert systems