PROLOG

Programming in Logic

ProLog

- **Pro**gramming in **Log**ic
- Based on the predicate logic
- A language in which the statements are logical expressions

Coding in ProLog

- Technically, you don't create a ProLog program
  - You create a ProLog database
- First step is to create a database using any ASCII text editor.
- A ProLog database file has a .pl extension

ProLog Components

- Names of constants and predicates begin with a lower case letter
- Variables begin with an upper case letter
- Entries always end with a dot
- Blank lines are OK
- Comments start with a percent (%) sign and stop at the end of the line

Facts

- A fact is a single piece of information
- Can be as simple as

  'It is raining today'. Or

  ahmed.
- In ProLog, facts are like the following:

  boy(jack).

  girl(jill).

  friends(jack, jill).

  plays(jack, hockey).

  go(jack, jill, 'up the hill').
- Facts with multiple arguments are called relations

Fact Interpretation

- ‘Intuitive’ interpretation

  eating(ana, burger).

  Intuitively means ‘Ana is eating burger’

  father(john, dean).

  Intuitively means ‘John is the father of Dean’

- Applying interpretation which makes more sense
- There should be consistency in interpretation

Rules

- Rules are used to express dependencies among facts
- These are used to generate new information from facts, other rules, and even themselves
- Examples:

```
child(X, Y) :- parent(Y, X).

odd(X) :- not even(X).

son(X, Y) :- parent(Y, X), male(X).

child(X, Y) :- son(X, Y); daughter(X, Y).
```

Logical Operators

- Connectives used in rules

ProLog	Read as	Logical operation
:-	IF	Implication
,	AND	Conjunction
;	OR	Disjunction
not	NOT	Negation

- In ProLog, it is left-side implication (←)
- \+ is another operator for NOT

Rules

- Form:

  head **:-** *body*.

- Example:

  grandparent(X, Z) :- parent(X, Y), parent(Y, Z).

  ancestor(X, Y) :- parent(Z, Y), ancestor(X, Z).

  - parent() clauses are called subgoals
  - First example is the same as the logical expression

$$\forall x \forall y \forall z \left(\left(P(x,y) \wedge P(y,z) \right) \rightarrow G(x,z) \right)$$

Writing ProLog database

- family.pl

```
parent(amy, bob).
parent(bob, cathy).
parent(bob, doug).
grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(Z, Y), parent(X, Z).
```

```
ancestor(X, Y) :- parent(X, Y); parent(Z, Y), parent(X, Z).
```

Using GProLog

- Use gprolog to query information from the database
- 'Compile' database file:

  ['*pl_filename*'].

  Make sure that the gprolog directory has been set to where the database file is
- Start querying when 'compile' has no errors

Using GProLog

- Try family:
  - Compile family

    ['family'].
  - Query family

    parent(amy, bob).

    - This results to true. This means that “amy is the parent of bob” is true.

    parent(bob, X).

    - This tries to extract all available information that matches the case in which bob is the parent.

    When query, gprolog may ask something like “true?” or “X = doug ?”

    When this happens, you may press “;” to allow gprolog to select the next possible solution (for variables, it will be the next applicable value), “a” to list all solutions, or RET (Enter key) to end/ accept the current solution.

ProLog! (family2.pl)

```
%start here

male(ali).

male(veli).

female(zeynep).

parent(ali, ayse).

parent(ali, ahmet).

parent(zeynep, ayse).
```

```
%continue here

father(X, Y) :- parent(X, Y), male(X).

mother(X, Y) :- parent(X, Y), female(X).

somebodysparent(X) :- father(X, Y);
                      mother(X, Y).

hasnochild(X) :- \+ parent(X, Y).
```

```
somebodysparent(X) :- father(X, _);
                      mother(X, _).
hasnochild(X) :- \+ parent(X, _).
```

ProLog Operators

Aside from the Logical Operators

Assignment

- Assign value to a variable using the **is** operator:

  Variable **is** *value*

- Examples:
  - X is 2
  - X is X+1

Arithmetic

- Arithmetic operators:

  + (addition)

  – (subtraction)

  * (multiplication)

  / (real division)

  // (integer division)

  mod (modulus)

  ** (power)

- Examples:
  - X+1
  - X+Y-Z

Relational

- Relation operators:

  = (equal)

  < (less than)

  > (greater than)

  =< (less than or equal)

  >= (greater than or equal)

  \= (not equal)

- Examples:
  - X < Y
  - X =< Y

Useful pre-defined ProLog elements

- rem(X, Y) – returns the remainder of X//Y
- write – displays its argument on screen