1.5 Introduction to Logic Programming (Prolog)

Simple example: Algorithm for checking ‘sorted list?’
We all know how to check if a list is sorted:

```
LET List_Aux = Input_list;
WHILE length (List_Aux) >= 2 DO BEGIN
    IF first(List_Aux) < first(rest(List_Aux)) THEN
        LET List_Aux = rest(List_Aux);
    ELSE
        RETURN false;
ENDWHILE
RETURN true;
```

• This algorithm solves the problem.
• Does the algorithm know how to solve the problem?
• Where is the knowledge about what to do to solve the problem?
1.5 Introduction to Logic Programming (Prolog)

The same problem: knowledge declaration for “sorted list?”
The following predicate logic rules declare what a sorted list is:

∀ l empty(l) → ordered(l) \hspace{1cm} (a)

∀ l, r rest(l, r) ∧ empty(r) → ordered(l) \hspace{1cm} (b)

∀ l, r, x, y rest(l, r) ∧ first(l, r) ∧ first(r, y)
∧ x ≤ y ∧ ordered(r) → ordered(l) \hspace{1cm} (c)

Interpretation:
- A list \( l \) is ordered if:
  - a) it has no elements, or
  - b) it has only one element, or
  - c) it has more than one element, and the first element is smaller or equal to the second, and the rest of the list is also ordered

1.5 Introduction to Logic Programming (Prolog)

The same problem: this time with Prolog

Compare the following Prolog program with the previous rules:

```
ordered([]).
ordered([X]).
ordered([X|Y|Zs]) :- X =< Y, ordered([Y|Zs]).
```

∀ l empty(l) → ordered(l)
∀ l, r rest(l, r) ∧ empty(r) → ordered(l)
∀ l, r, x, y rest(l, r) ∧ first(l, r) ∧ first(r, y)
∧ x ≤ y ∧ ordered(r) → ordered(l)

- They are almost identical, except for the syntax of Prolog
- But in addition this is a runnable program!
- Then, where it is the algorithm?
- It is clear what it does, but not so much how it does.
Let’s see “how” Prolog runs it: Unification, Resolution, ...

1. Let \( Q \) be a given query, \( \text{ordered([4,7])} \)
2. Since \( Q \) does not unify with either of the first two rules, we try the third one, and perform a unification of \( \text{ordered([4,7])} \) with \( \text{ordered([X|[Y|Zs]])} \), resulting in: \( X=4, \ Y=7, \ Zs=[] \)
3. That allows us to proceed, and verify the conditions of the third rule:
   a) \( X < Y \) which instantiates as \( 4 < 7 \), so it resolves as true,
   b) \( \text{ordered([7])} \) which is treated as a (recursive) new query \( Q^\prime \) and Prolog attempts to solve it in the same way ...
4. After checking that \( Q^\prime \) holds true (unifying with second rule), and given that \( Q^\prime \) was the last condition required to apply the third rule, we conclude that the initial query "logically follows" from the given program/rules.

```
ordered([]).
ordered([X]).
ordered([X|[Y|Zs]]):- X =< Y, ordered([Y|Zs]).
```

### Prolog's syntax and terminology

- **A prolog program consists of a set of facts and rules:**
  
  ```prolog
  likes(john, mary).
  likes(mary, peter).
  likes(mary, X) :- hates(john, X).
  ```
1.5 Introduction to Logic Programming (Prolog)

Given a set of facts and rules, questions can be asked (e.g. at the console):

Database:

```
likes(john, mary).
likes(mary, peter).
likes(mary, X) :- hates(john, X).
```

Questions:

```
?- likes(john, peter).
no
?- hates(john, peter).
yes
```
1.5 Introduction to Logic Programming (Prolog)

Variables
- Variables are logical variables, not memory cells holding contents.
- If a unification binds a variable to a specific value, the same variable cannot be bound to a different value: $p(X, X)$ does not unify with $p(3, 2)$.
- The anonymous variable is _ (the only variable that unifies with anything and does not remember its binding).

Values
- The arguments to a predicate are either constants, variables, or lists.
  (structures also, but ignored here)
- List: loves(john, [mary, sara, louisa])
- There are no functions in prolog (except built-in operators such as addition).
- Thus cannot write related(2, 3, f(2, 3)) where f() is expected to return a value.

Expressions
- Some built in predicates defined for mathematics, e.g., $+(B, 1, X)$.
- Rather than matching these predicates against the fact-base, these are evaluated specially.
- Prolog allows these predicates to be written differently: $X = B + 1$
Lists
• Formed with [], and commas and |
• Lists can be nested: 
  [ [a, 1], [b, 2], [c, 3] ]
• Lists can be arguments of predicates: sorted([3, 5, 9])
• Lists can contain variable to be unified
  symmetric([X, _, Y])
• ‘Bar’ syntax: \([A \mid [B \ C]]\) is equivalent to \([A \ B \ C]\)
  – The item before the bar is the HEAD of the list (‘car’ in Lisp)
  – The item after the bar is the TAIL of the list (the ‘cdr’ in Lisp)
• The following are equivalent:
  \([a, b, c]\) \([a \mid [b, c]]\)
  \([c]\) \([c \mid [\ ]]\)

Example: Family

Database:
  father(luis, maria).
  mother(rosa, maria).
  mother(rosa, pedro).
Questions:
  ? father(luis, X)
    X = maria
  ? mother(rosa, X)
    X = maria
  ;
    X = pedro
  ;
    no

Prolog will give just one answer to a query. Typing ; gives the next, until no other answers remain
1.5 Introduction to Logic Programming (Prolog)

Rules
• Prolog rules are Horn Clauses:
  – The conclusion/consequent is written first,
  – Followed by the logic implication symbol, in Prolog :- ,
  – Followed by the antecedents/premises separated by commas (conjunction in Prolog)

```prolog
grandfather(X,Y) :- father(X,Z), father(Z,Y).
```

- Consequent - A term which is established if the RHS of the rule is established.
- Implication Symbol
- Premises - Each one is a prolog term.
  - If one term fails to resolve, processing stops, and the rule fails.
  - Variables established in one term passed to the next

Rules: Example 1

```prolog
grandfather(X,Y) :- father(X,Z), father(Z,Y).
inheritor(X,Y) :- father(Y,X).
inheritor(X,Y) :- mother(Y,X).
```

- **Conjunction**: the comma between two premises. *A if B and C*
- **Disjunction**: the availability of alternative rules:
  – A if B, A if C → A if B or C
- **Negation**: built-in predicate ‘not’:
  ```prolog
  father(X,Y) :- parent(X,Y), not(mother(X,Y)).
  ```
1.5 Introduction to Logic Programming (Prolog)

Rules: Example 2

father(luis, juan).
father(luis, maria).
mother(rosa, maria).
mother(rosa, pedro).
mother(maria, oscar).
grandfather(X, Y) :- father(X, Z), father(Z, Y).
grandfather(X, Y) :- father(X, Z), mother(Z, Y).
inheritor(X, Y) :- father(Y, X).
inheritor(X, Y) :- mother(Y, X).

? inheritor(luis, juan)
? inheritor(maria, Y)
? inheritor(X, maria)
? grandfather(luis, N)
? grandfather(A, oscar)
? grandfather(A, N)

1.5 Introduction to Logic Programming (Prolog)

Prolog Programs with Facts and Rules to develop in class

- **member**(A, B): returns yes if A in list B.
- **nmember**(A, B): as with member, but also test for A in a nested list within B
- **llen**(A, B): A is set to length of list B
- **remove**(A, B): removes first occurrence of A from list B
- **nremove**(A, B): removes all occurrences of A from list B or any nested list in B.

• You can try simple programs and queries interactively using
• We will come back to Prolog later (in section 3) ...