# Introduction to Prolog

CSCI3180 Principles of Programming Languages

Tutorial 9

# Imperative vs Declarative

- Develop a program to solve a probelm
  - What the problem is
  - How to solve the problem
- Imperative
  - How to solve the problem
  - Take longer time to develop a program
- Declarative
  - Do not worry about how to solve the problem
  - What the problem is
  - Less time for program development

# Prolog

- Prolog programming is divided into two stages.
  - Asserting what is true (building a program).
    - Facts
    - Rules
  - Asking what you want to know (running a program).
    - Query

# SICStus Prolog

- You are supposed to test your work on the machine "solar1.cse.cuhk.edu.hk" under SICStus 3.12.7, by directly typing "sics".

```
sparc1.cs.cuhk.hk:/uac/gds/cyhong> sics
SICStus 3.12.7 (sparc-solaris-5.7): Fri Oct  6 00:12:30 MET DST 2006
Licensed to cse.cuhk.edu.hk
| ?-
```

- Prolog Prompt: | ?-
- To load a file containing Prolog code

```
$ sics
| ?- ['asg2.pl'].
| ?- consult('asg2.pl').
| ?- reconsult('asg2.pl').
```

Alternatively…

```
| ?- [asg2].
| ?- consult(asg2).
| ?- reconsult(asg2).
```

# Example: Basic Idea

- Knowledge Base

```
likes(peter,mary).
likes(may,sam).
likes(sam,may).
likes(mary,sam).
canMarry(X,Y) :- likes(X,Y), likes(Y,X).
```

Facts

Rule

- Queries

```
| ?- canMarry(May,sam).
yes
| ?- canMarry(peter,mary).
no
| ?- canMarry(X, may).
X = sam.
| ?- canMarry(X, Y).
```

# Anonymous Variables

- A variable begin with *underscore _*

- Knowledge Base:

  likes(peter,mary).

  likes(peter,alice).

```
| ?- likes(peter,Who).
Who = mary ? ;
Who = alice ? ;
no
| ?- likes(peter,_who).
yes
| ?- likes(peter,_).
yes
| ?- likes(john,_who).
no
| ?-
```

Who does Peter like?

Does Peter like anybody?

# Syntax

- Finite set of clauses
- Clause (Rules) : Head and Body

  Syntax: $A$ :- $B_1$, $B_2$, $B_3$ .

- Fact: Clause without Body and variables

  Syntax: A.

- likes(peter, X) is a rule
- Comments:

  % comment line

  /* comment block */

# Syntax

- Rules

  Syntax: $A \text{ :- } A_1, A_2.$

IF

AND

- Rule Ordering
  - Match sequentially, from top to down

$A \text{ :- } A_1, A_2 .$

$A \text{ :- } B_1, B_2 .$

The first one is matched first, and then the second rule

- Goal Ordering
  - Ordering of terms within the body of a rule
  - Answered sequentially, from left to right

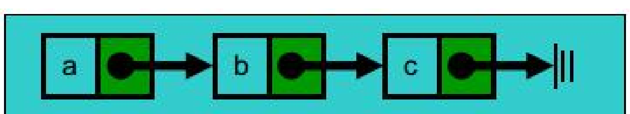
$A \text{ :- } A_1, A_2 .$

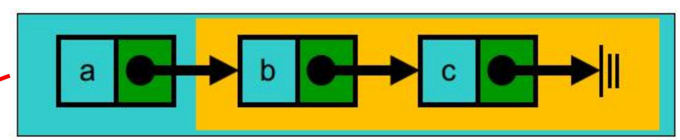
$A_1$ answered first and then $A_2$

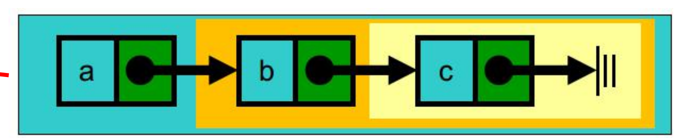
# Lists

- Prolog's built-in list representation
- $[]$ denote empty list
- Syntactic sugar: $[X|Y]$
  - $X$ is the head and $Y$ is the tail of list
- Example: a list (a, b, c)

$$
\begin{aligned}
[a,b,c] &= [a,b|[c]] \\
&= [a|[b,c]] \\
&= [a|[b|[c|[]]]]
\end{aligned}
$$



# append/3

```
append([], L, L).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).
```

| | L1 | | L2 | | L3 |
|---|---|---|---|---|---|
| If : | [3, 4] | + | [5, 6, 7] | = | [3, 4, 5, 6, 7] |

Then: [ 2 , 3 , 4 ] + [ 5 , 6 , 7 ]

= [ 2 , 3 , 4 , 5 , 6 , 7 ]

# append/3

```
append([], L, L).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).
```

List 1

$[x_1, x_2, \ldots, x_n]$

List 2

$[y_1, y_2, \ldots, y_n]$

List 3

$[x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n]$

# append/3

```
append([], L, L).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).
```

List 1 — L1: $[x_1, \boxed{x_2, \ldots, x_n]}$

List 2 — L2: $\boxed{[y_1, y_2, \ldots, y_n]}$

X

List 3 — L3: $[x_1, \boxed{x_2, \ldots, x_n, y_1, y_2, \ldots, y_n]}$

# append/3

```
append([], L, L).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).
```

List 1

L1

$[x_1, \boxed{x_2, \ldots, x_n]}$

List 2

L2

$\boxed{[y_1, y_2, \ldots, y_n]}$

List 3

$[x_1, \boxed{x_2, \ldots, x_n, y_1, y_2, \ldots, y_n]}$

L3

# append/3

```
append([], L, L).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).
```

List 1

L1

$[x_1, x_2, \boxed{\ldots, x_n]}$

List 2

L2

$\boxed{[y_1, y_2, \ldots, y_n]}$

X

List 3

$[x_1, x_2, \boxed{\ldots, x_n, y_1, y_2, \ldots, y_n]}$

L3

# append/3

```
append([], L, L).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).
```

List 1

[ $x_1$, $x_2$, …, $x_n$ ]

List 2

L

[ $y_1$, $y_2$, …, $y_n$ ]

List 3

[ $x_1$, $x_2$, …, $x_n$, $y_1$, $y_2$, …, $y_n$ ]

L

It is true !

# append/3

```
append([], L, L).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).
```

List 1 — L1: $[x_1, x_2, \boxed{\dots, x_n]}$

List 2 — L2: $\boxed{[y_1, y_2, \dots, y_n]}$

List 3: $[x_1, x_2, \boxed{\dots, x_n, y_1, y_2, \dots, y_n]}$ — L3

It is true !

# append/3

```
append([], L, L).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).
```

List 1

$[x_1, x_2, \ldots, x_n]$

List 2

$[y_1, y_2, \ldots, y_n]$

List 3

$[x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n]$

It is true !

# uses of append/3

```
append([], L, L).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).
```

- Check concatenation

```
| ?- append([1,2],[3,4],[1,2,3,4]).
| ?- append([1,2],[4,5],[1,2,3,4]).
```

true

false

- Concatenate 2 lists

```
| ?- append([1,2],[3,4],L).
L = [1,2,3,4] ? ;
no
```

# uses of append/3

```
append([], L, L).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).
```

- Check head element
  - ?- append([1],_,[1,2,3,4]).
  - ?- append([2],_,[1,2,3,4]).
- Check last element
  - ?- append(_,[4],[1,2,3,4]).
  - ?- append(_,[1],[1,2,3,4]).

# uses of append/3

```
append([], L, L).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).
```

- Find head element

```
?- append([H],_,[1,2,3,4]).
```

- Find last element

```
?- append(_,[E],[1,2,3,4]).
```

# uses of append/3

```
append([], L, L).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).
```

- Subtract last part of a list

```
| ?- append(L,[3,4],[1,2,3,4]).
```

- Subtract first part of a list

```
| ?- append([1,2],L,[1,2,3,4]).
```

- Decompose a list

```
| ?- append(L1,L2,[1,2,3,4]).
```

```
L1 = [],
L2 = [1,2,3,4] ? ;
L1 = [1],
L2 = [2,3,4] ? ;
L1 = [1,2],
L2 = [3,4] ? ;
L1 = [1,2,3],
L2 = [4] ? ;
L1 = [1,2,3,4],
L2 = [] ? ;
no
```

- more ...

# Assignment 4 Q1 - list operation

- element_last(X, L)
  - which is true if the last element in list L is X
  - ?- element_last(e, [a,b,c,d,e]).
  - True

# Q1 - list operation

- element_n(X, L, N)
  - which is true if the Nth element in list L is X.
  - | ?- elementn(c, [a,b,c,d,e], s(s(s(0)))).
  - True

# Q1 - list operation

- remove_n(X, L1, N, L2)
  - get the resulting list L2 that is obtained from L1 by removing the N th element, as well as get the removed element X.
  - | ?- remove_n(X, [a,b,c,d,e], s(s(s(0))), L2).
  - X = c,
    L2 = [a, b, d, e].

# Q1 - list operation

- remove_n(X, L1, N, L2)
  - get the resulting list L2 that is obtained from L1 by removing the N th element, as well as get the removed element X.
  - | ?- remove_n(X, [a,b,c,d,e], s(s(s(0))), L2).
  - X = c,

    L2 = [a, b, d, e].

# Q1 - list operation

- insert_n(X, L1, N, L2)
  - get the resulting list L2 that is obtained by inserting X to the position before Nth element of list L1.
  - | ?- insertn(h, [a,b,c], s(s(0)), L2).
  - L2 = [a,h,b,c]

do it with remove_n !

# Q1 - list operation

- repeat_three(L1, L2)
  - get the resulting list L2 that repeats each element inlist L1 for three times.
  - | ?- repeat_three([a,b,c,d,e],X).
  - X = [a,a,a,b,b,b,c,c,c,d,d,d,e,e,e]

# Q2 - Multiway tree

- A multi-way tree is composed of a root and a sequence of sub-trees (children), which are multi-way tree themselves. The list of sub-trees of a certain node is also called a forest.

- In Prolog, we represent a multi-way tree by the term “mt(X, F)”, where X denotes the rootand F denotes the forest of sub-trees.

# Q2 - Multiway tree

- Represent the multiway tree shown in Figure 1 as a Prolog term.
  - (Hint: represent forest as a list of multiway trees).

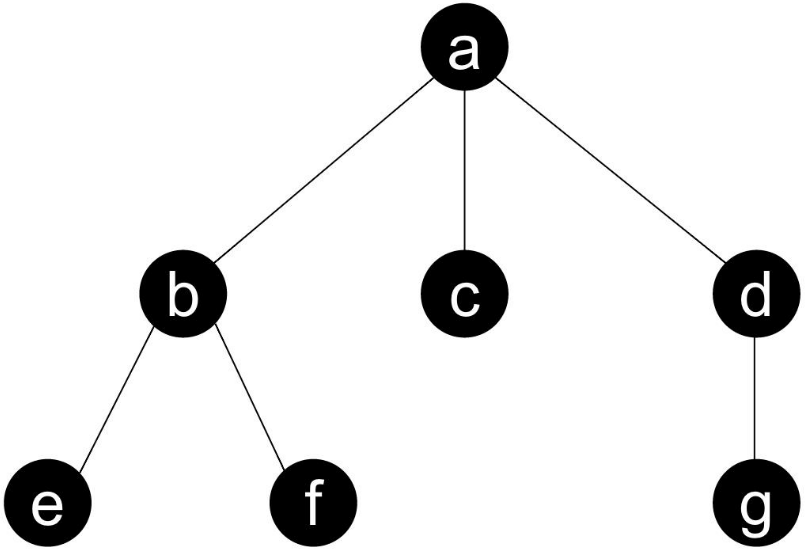



Figure 1: An example of multi-way tree

# Q2 - Multiway tree

- Define the predicate *is_tree(Term)*
- Define the predicate *num_node(Tree, N)*
- Define *sum_length(Tree, L)*
  - which is true if "L" is the sum of lengths of all internal paths in "Tree".

# Q2 - Multiway tree

- *sum_length(Tree, L)*
  - The length of an internal path from the root node r to an internal node n is the distance from r to n.
  - *L = 1 (b) + 1 (c) + 1 (d) + 2 (e) + 2 (f) + 2 (g) = 9*

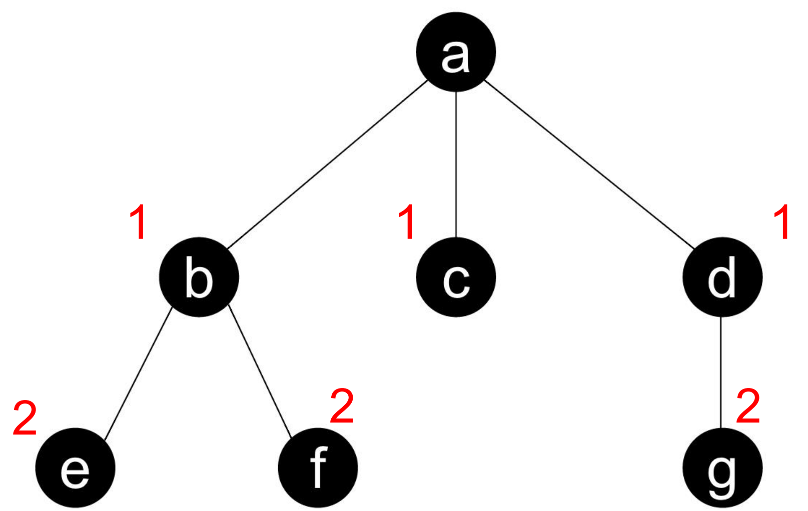



Figure 1: An example of multi-way tree