TUTORIAL 5

CSCI3230 (2013-2014 First Term)

By Paco WONG (pkwong@cse.cuhk.edu.hk)

Outline

- Build-in Predicates
- Control the flow of satisfaction
- Examples

Built-in Predicates

- Most Prolog systems provide many built-in predicates and functions such as
  - Arithmetic functions (+, -, mod, is, sin, cos, floor, exp, …)
  - Bit-wise operations (/\, \/, \, <<, >>, xor)
  - Term comparison (==, \==, @<, @>, …)
  - Input/Output (read, write, nl, …)
  - Control
  - Meta-logical
  - …

For more http://www.swi-prolog.org/pldoc/doc_for?object=section(1,'4',swi('/doc/Manual/builtin.html'))

Built-in Predicates

Example 1

```
?- X is sin(2*pi). %Built-in constant and function sin
X = -2.4492127076447545e-16.
?- X is 4 >> 1.
X = 2
?- Y @< b. %Variables<Numbers<Atoms<Strings<Compound Terms
true.
?- read(X),read(Y),Z is cos(X*Y),write('Complete').
|: 2.                  %For each term, you need to add a .
|: pi.
Complete
X = 2,                 %Number 2 has been unified with X
Y = pi,
Z = 1.0.
```

Equivalence

Operator	Meaning	Description
TermA == TermB	Testing for equivalence	A variable is only identical to a sharing variable.
TermA =@= TermB	Testing for a variant (or structurally equivalence)	True iff there exists a renaming of the variables in A that makes A equivalent (==) to B and vice versa.
TermA = TermB	Testing for unification	True if the unification succeeds, and the terms in A and B will be unified.
TermA is TermB	Testing for numerical value	True if both terms has the same numerical value after evaluation of TermB.

Equivalence

Example 2-1

```
?- X is 6+3, S is 9.
X = 9,
S = 9.
?- X is 9, S is 10, X is S.
false.
?- 1+3 is 1+3.
false. %Evaluation only on the last term of is
```

Example 2-2

```
?- f(A,B) == f(A,B).
true.
?- f(A,B) == f(X,Y).
false.
?- f(A,B) =@= f(X,Y).
true.  %Renaming X to A, Y to B
?- f1(A,f2(B)) =@= f1(_,f2(C)).
true.  %_ is anonymous variable
```

Example 2-3

```
?- f(A,B) =@= f(A,b).
false. %B is var, b is atom
?- f(A,B) = f(a,b).
A = a,
B = b. %Can be unified
```

Assert and Retract

- Modify a (running) program *during execution*
  - NOT encouraged unless you have some good reasons, e.g. memoization.
- **ASSERT** to insert a fact or rule
- **RETRACT** to remove a fact or rule
  - Abolish is evil.

Example 3

```
?- assert(color(apple,red)).
true.
?- color(apple,red).
true.
```

For more http://www.swi-prolog.org/pldoc/doc_for?object=section(2,'4.13',swi('/doc/Manual/db.html'))

Control the Flow of Satisfaction

- The semantics of Prolog programs does not care about order
- Conjunction is commutative
  - E.g. `P :- Q,R,S.` should mean the same as `P :- S,R,Q.` logically
- In practice
  - the order matters
  - side effects are involved
  - most Prolog systems use left to right DFS, top to bottom order
- To control the order of matching for query
  - Place the facts and rules in a suitable sequence
  - Use **!** and **fail** operator

Recap: Backtracking

- When asked $P_1(..), P_2(..), \ldots, P_n(..)$.
  - If anyone fails (due to instantiation), say P_i, Prolog backtracks, and try an alternative of P_{i-1}
- After a successful query,
  - If user presses '**;**', backtrack and try alternatives.

Tutorial 4: Example 6

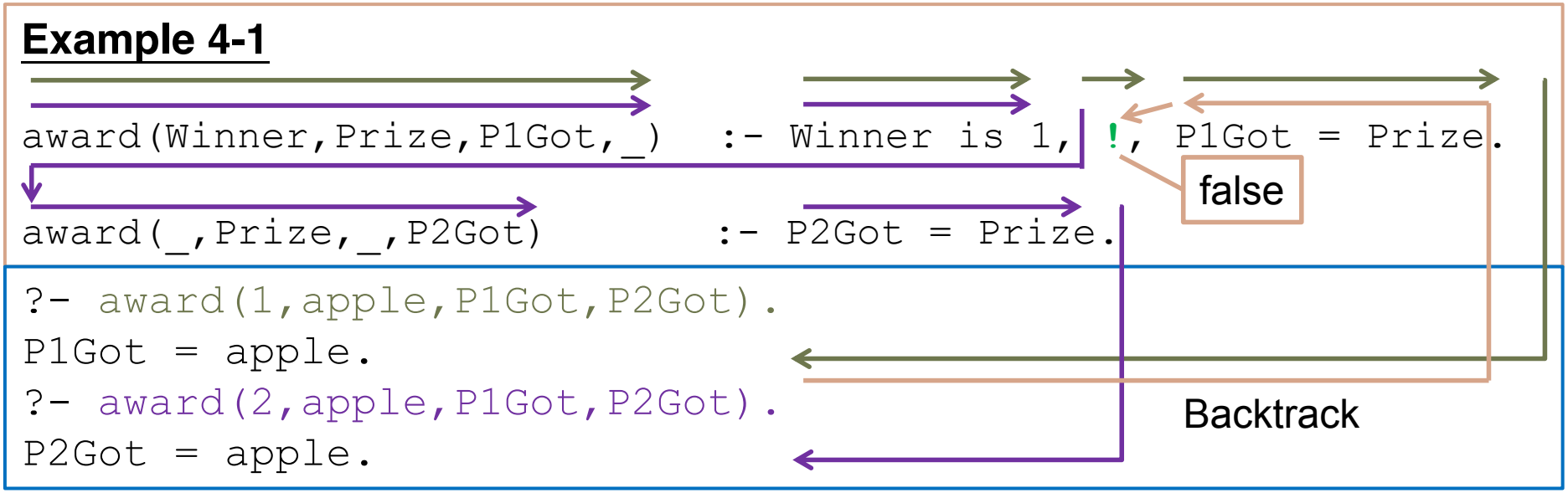
```
likes(mary,donut). %Fact 1
likes(mary,froyo). %Fact 2
likes(kate,froyo). %Fact 3
```

```
?- likes(mary,F),likes(kate,F). %Sth both Mary and Kate like
F = froyo.
```

Cut !

- **!** is used for search control
  - When it is first encountered as a goal, it succeeds
  - If backtracking returns to the cut, it fails the parent-goal (Head of the rule)
  - Reduce memory usage as less backtracking points are stored

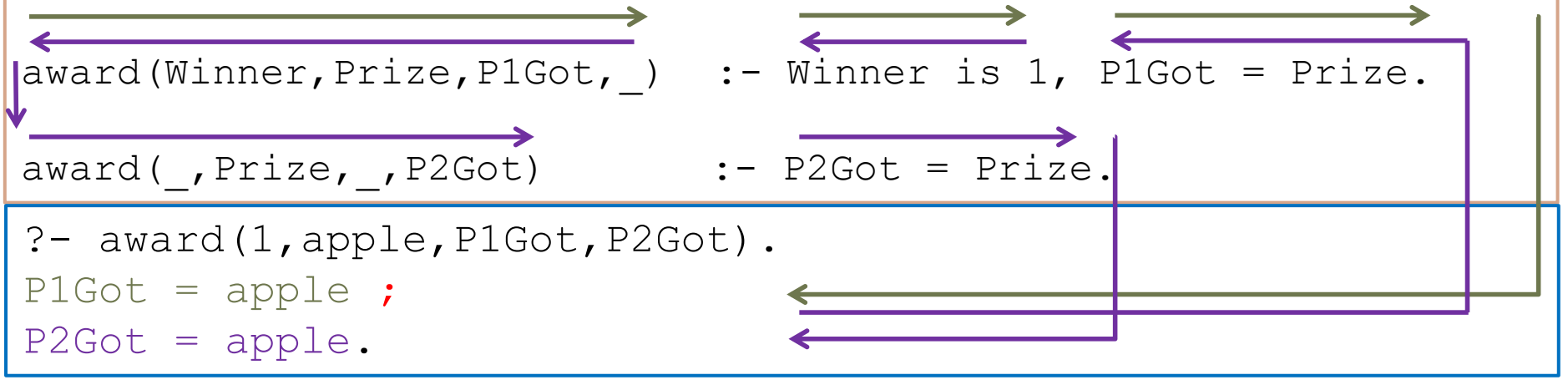
Example 4-1



Without Cut

Example 4-2 (Remove !)

```
award(Winner,Prize,P1Got,_)   :- Winner is 1, P1Got = Prize.
award(_,Prize,_,P2Got)        :- P2Got = Prize.
```



```
?- award(1,apple,P1Got,P2Got).
P1Got = apple ;
P2Got = apple.
```

Visualizing Cut: Code View

false

```
p:-P01(..),P02(..),…,P0i(..),…,P0n(..).
p:-P11(..),P12(..),…,P1i(..),!,…,P1n(..).
p:-P21(..),P22(..),…,P2i(..),…,P2n(..).
...

p:-P01(..),P02(..),…,P0i(..),…,P0n(..).
p:-P11(..),P12(..),…,P1i(..),!,…,P1j(..),…,P1n(..).
p:-P21(..),P22(..),…,P2i(..),…,P2n(..).
...

p:-P01(..),P02(..),…,P0i(..),…,P0n(..).
p:-P11(..),P12(..),…,!,…,P1j-1(..),P1j(..),…,P1n(..).
p:-P21(..),P22(..),…,P2i(..),…,P2n(..).
...
```

Ignore these predicates

false

Fail

- **FAIL** is a predicate which is always false.
- **\+** tells whether the predicate is NOT provable. It is defined as if by

```
\+(P) :- P, !, fail.
\+(P).
```

Example 5

```
illegal(X,Y) :- X=Y, !, fail.
illegal(X,Y). %Is illegal iff X and Y cannot be unified
```

```
?- illegal(a,b).
true.
?- illegal(a,B).
false. %NOT illegal, Atom a can be unified with variable B
```

findall(Object,Goal,List).

- Produces a list *List* of all the objects *Object* that satisfy the goal *Goal*.

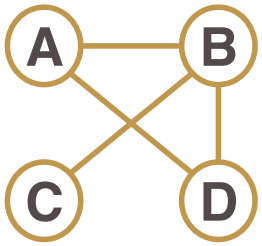
Example 6

```
dessert(froyo).
dessert(lava_cake).
dessert(marble_cake).
likes(mary,froyo).
likes(mary,lava_cake).
likes(mary,banana).
likes(kate,froyo).
likes(kate,marble_cake).
likes_dessert(P,F):-dessert(F),likes(P,F).
```

```
?- findall(F,likes_dessert(mary,F),L).
L = [froyo, lava_cake].
?- findall(F,likes_dessert(P,F),L).
L = [froyo, froyo, lava_cake, marble_cake].
```

EXAMPLES

Example: Links in Graph




```
1.  link(a,b).
2.  link(b,c).
3.  link(a,d).
4.  link(b,d).
5.  link(X,Y):-
    link(X,Z),link(Z,Y).
```

```
?- link(a,K).
K = b ;
K = d ;
K = c ;
K = d ;
ERROR: Out of local stack
```

In our usage of *link*, it always matches to the fifth rule, which means there is no base case.

Explanation

link(a,K).

1. Matches 1, Return K=b, Press ;
2. Matches 3, Return K=d, Press ;
3. Match 5

New sub-goal: link(a,Z), link(Z,K).

1. link(a,Z) matches 1, unified Z to b.
2. link(b,K) matches 2. Return K=c., Press ;
3. link(b,K) matches 4. Return K=d., Press ;

New sub-goal: link(b,Z), link(Z,K).

1. link(b,Z) matches 2, unified Z to c.
2. link(c,K) matches 5,

New sub-goal: link(c,Z),link(Z,K).

1. link(c,Z) matches 5. (let Z be Z_{old})

New sub-goal: link(c,Z),link(Z,Z_{old}).

1. link(c,Z) matches 5.

…

(loop forever)

Renaming facts and rules

```
1.  link(a,b).
2.  link(b,c).
3.  link(a,d).
4.  link(b,d).
5.  path(X,Y):-link(X,Y);link(Y,X). %Single hop
6.  path(X,Y):-link(X,Z),link(Z,Y). %More than one hop
7.  path(X,Y):-link(Z,X),link(Z,Y).
8.  path(X,Y):-link(X,Z),link(Y,Z).
9.  path(X,Y):-link(Z,X),link(Y,Z).
```

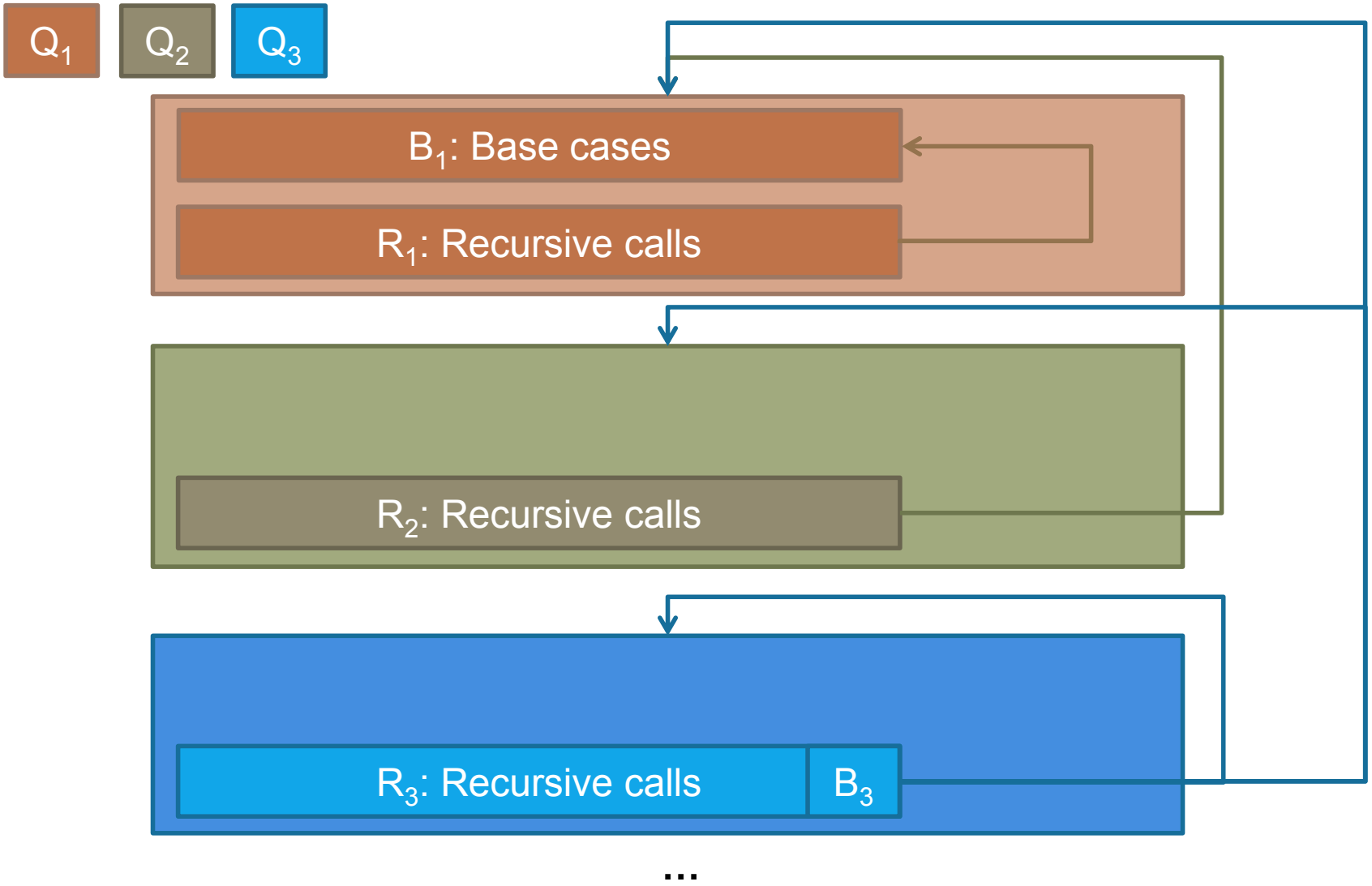
```
1.  link(a,b).
2.  link(b,c).
3.  link(a,d).
4.  link(b,d).
5.  edge(X,Y):-link(X,Y);link(Y,X).
6.  path(X,Y):-edge(X,Y).                    %Single hop
7.  path(X,Y):-edge(X,Z),edge(Y,Z).          %More than one hop
```

Avoid unification between facts and rules

- Name differently
- Use different terms

Handle the base case in the rules using AND.

Recursion, Unification and Ordering



Example: Membership

- Define `member(X,Y)` to be true iff `X` (a term) is a member of the list `Y`
- NO need to explicitly check for empty list
  - because an empty list cannot be unified with the clauses
  - so `member(a,[])` is automatically not provable

Example 6

```
member(X,[X|_]). %Recall [a] is equivalent to [a|[]]
member(X,[_|T]) :- member(X,T).
```

```
?- member(s,[f,i,s,h]).
true ;
?- member(X,[f,i,s,h]).
X = f ;
X = i ;
X = s ;
X = h.
```

Example: Membership

Example 6

```
member(X,[X|_]).
member(X,[_|T]) :- member(X,T).
```

```
?- member(X,[f,i,s,h]).
```

```
member(X,[X|_]).
member(X,[_|T]) :- member(X,T).
```

```
member(X,[f|[i,s,h]])    X=f
member(X,[f|[i,s,h]]) T=[i,s,h]
```

New sub-goal: `member(X,[i,s,h])`

```
member(X,[X|_]).
member(X,[_|T]) :- member(X,T).
```

```
member(X,[i|[s,h]])      X=i
member(X,[i|[s,h]])   T=[s,h]
```

New sub-goal: `member(X,[s,h])`

```
X=s
```

New sub-goal: `member(X,[h])`

```
T=[h]
```

```
member(X,[X|_]).
member(X,[_|T]) :- member(X,T).
```

```
member(X,[h|[]])         X=h
member(X,[h|[]])         T=[]
```

New sub-goal: `member(X,[])`

```
false
```

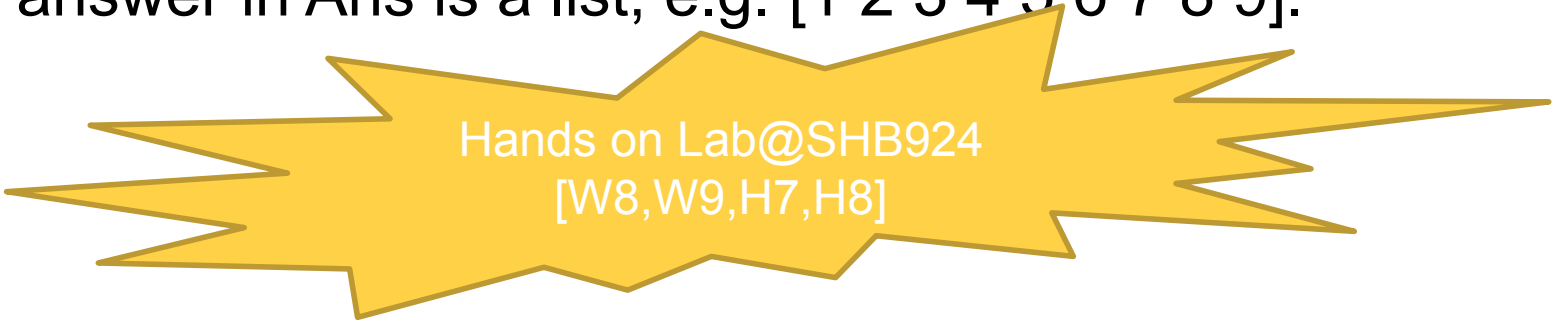
Summary

- Build-in Predicates
  - ==, =@=, …
- Flow of Satisfaction
  - Order
  - Cut !
  - Fail
- findall/3
- Examples
  - Unification between facts and rules

Feedback: email or http://goo.gl/5VLYA

Try it yourself

- Given a list L of integer, write *findmax(L,Ans)* to find the largest one and stored it in Ans.
- Tower of Hanoi: Move N disks from the left peg to the right peg using the center peg as an auxiliary holding peg. At no time can a larger disk be placed upon a smaller disk. Write *hanoi(N)*, where N is the number of disks on the left peg, to produce a series of instructions, e.g. “move a disk from left to middle”.
- Fill in a 3x3 grid with number from 1-9 with each number appearing once only. Write a *puzzle3x3(Ans)* to do this. The answer in Ans is a list, e.g. [1 2 3 4 5 6 7 8 9].



Announcement

- Written Assignment 2
- Prolog Assignment

Printing a matrix

- /* print_board([[a,b,c,d],[e,f,g,h]]). */
- /* Utility */
- extra_space(w,' ').
- extra_space(b,' ').
- extra_space(m,' ').
- extra_space(s,' ').
- extra_space(_,'').

- print_row([]).
- print_row([H|T]):-write(H),extra_space(H,X),write(X),write(' '),print_row(T).
- print_board([]).
- print_board([H|T]):-
- print_row(H),write('\n'),print_board(T).

Reference

- Reference manual of SWI-Prolog
  - http://www.swi-prolog.org/pldoc/refman/
- More advanced Prolog
  - The Craft of Prolog by Richard A. O'Keefe
- A debug technique
  - http://stackoverflow.com/questions/13111591/prolog-check-if-two-lists-have-the-same-elements