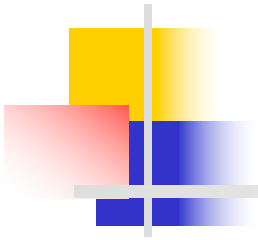
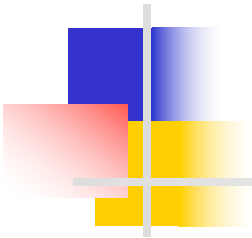
COMP3411-9814- Artificial Intelligence



Prolog
Syntax, Lists, Operatores & Arithmetic

2020 – Summer Term

Tatjana Zrimec

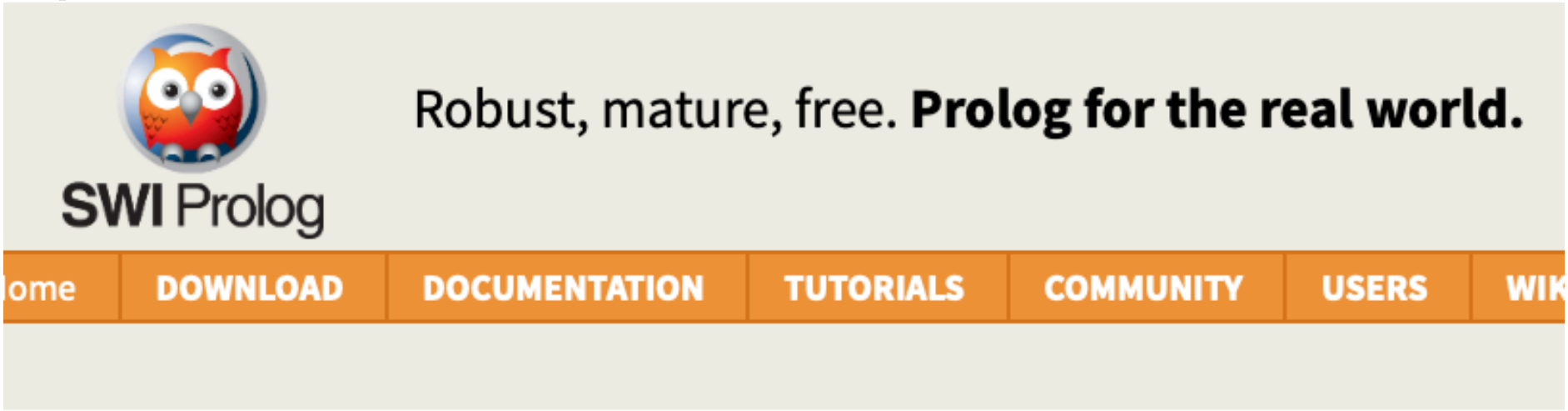


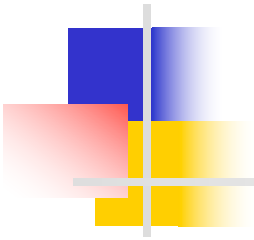
SWI Prolog

- SWI-Prolog
- http://www.swi-prolog.org

- SWI-Prolog reference manual
- http://www.swi-prolog.org/pldoc/doc_for?object=manual

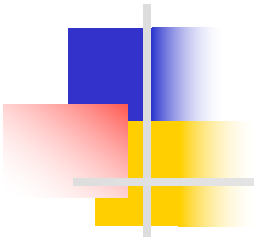
SWI Prolog


SWI Prolog
Robust, mature, free. Prolog for the real world.
ome
DOWNLOAD
DOCUMENTATION
TUTORIALS
COMMUNITY
USERS
WIK
-Prolog offers a comprehensive free Prolog environment. Since its start in
7, SWI-Prolog development has been driven by the needs of real world
lications. SWI-Prolog is widely used in research and education as well as
mercial applications. Join over a million users who have downloaded SW
og. more ...



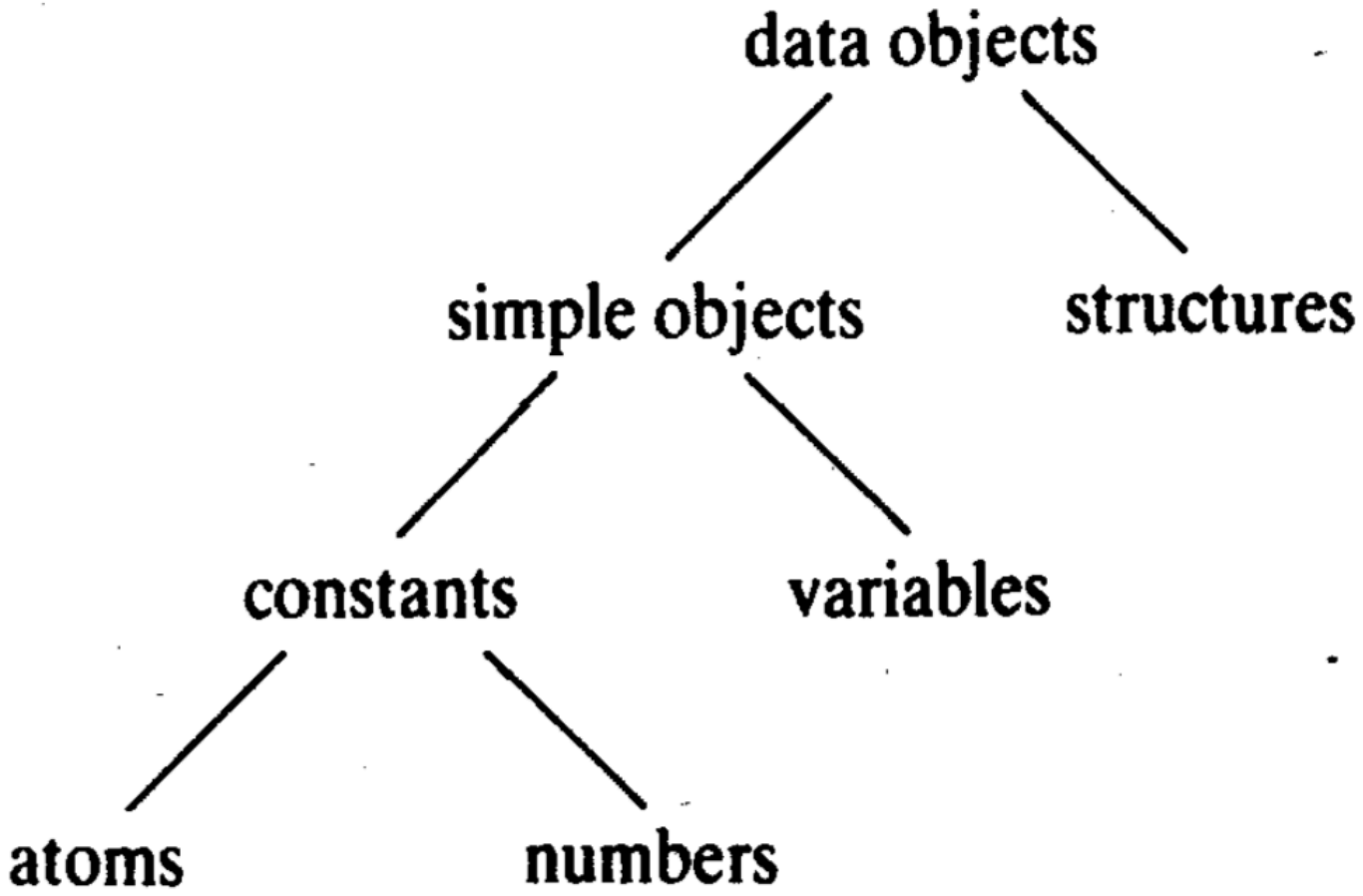
Outline

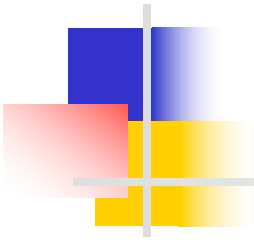
- Syntax and semantics
- Data objects
- Structures
- Matching
- Lists
- Operators
- Arithmetic



Prolog – syntax and semantics

Data objects in Prolog





Object Syntax

- The type of object is always recognizable from a syntactic form

Three Syntactic Forms for Atoms

(1) Strings of letters, digits and the underscore character "-", starting with lower case letter

x x15 x_15 aBC_CBa7

alpha_beta_algorithm taxi_35

peter missJones miss_Jones2

Three Syntactic Forms for Atoms

(2) Strings of special characters

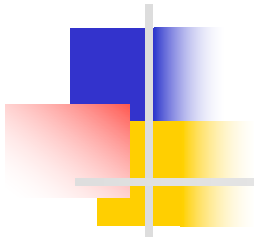
---> <==> <<

. < > + ++ ! .. .::. ::= []

(3) Strings of characters enclosed in single quotes

‘X_35’ ‘Peter’ ‘Britney Spears’

This is useful if we want an atom to start with a capital letter



Numbers

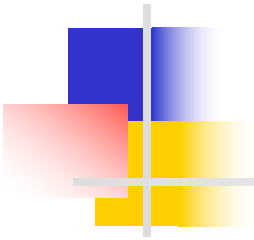
- Strings of special characters

  1 1313 0 -55

- Real numbers

  3.14 -0.0045 1.34E-21 1.34e-21

Real numbers not much used in Prolog



Variables

- Variable are strings of letters, digits and underscore character :

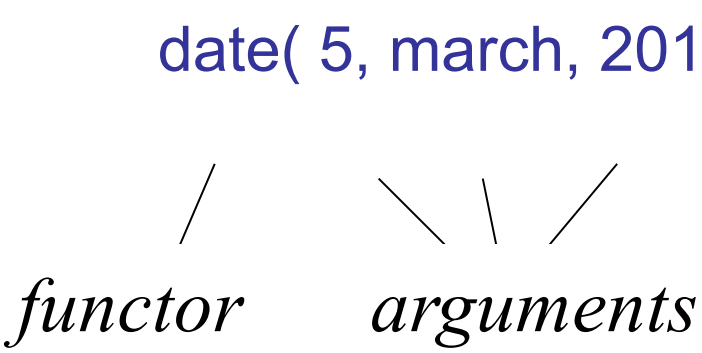
  X Results Object2B Participant_list

  _x35 _335

- The lexical range of variable names is one clause.

Structures

- Structures are multi-component objects
  - For example, a date is a three-component structure
  - Date March 5 2017:

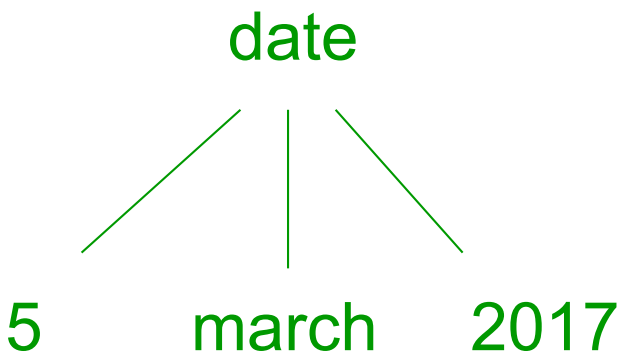


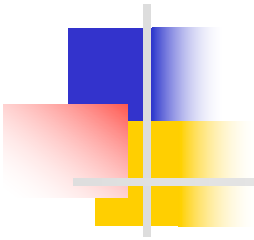
- The argument can be any object, including the structure

Tree representation of structures

- Structures are sometimes illustrated as trees:

date(5, march, 2017)

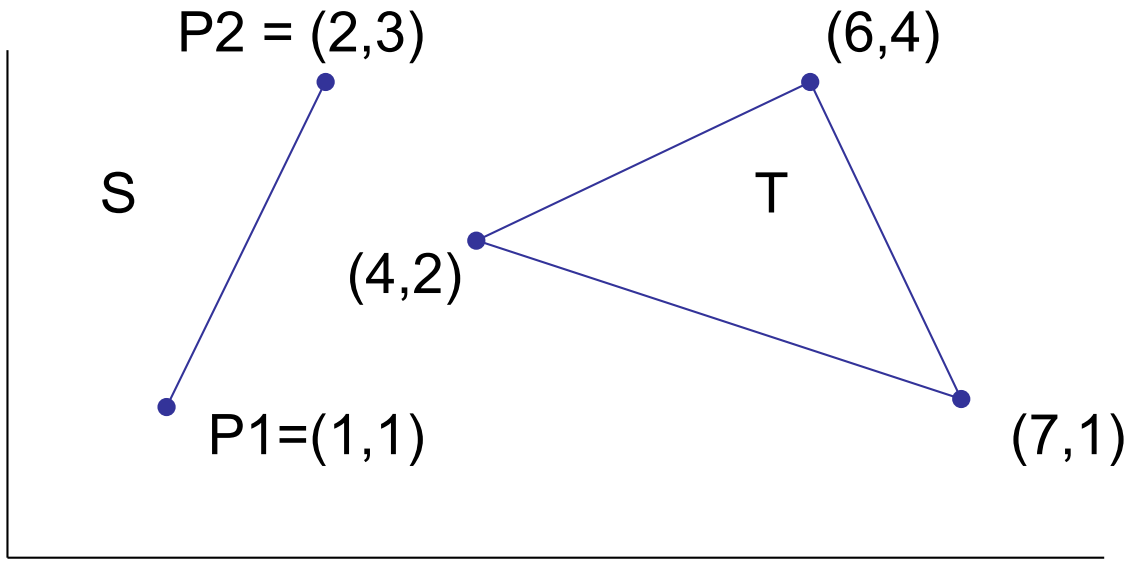




Structure

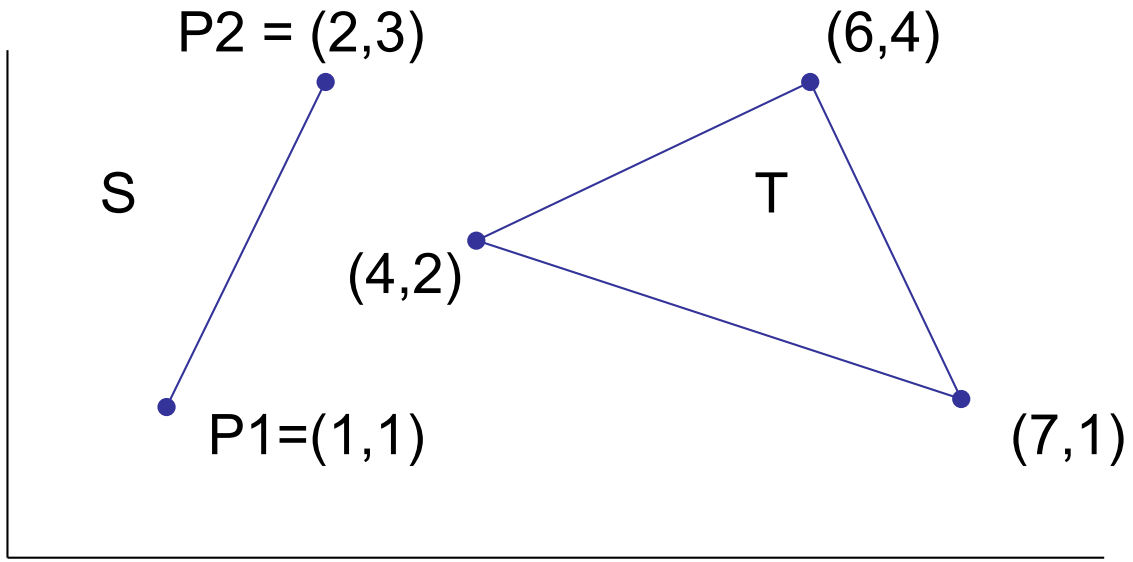
- Structured objects are objects that several components.
  - The components can be also structures.
- All structured objects in the prolog can be illustrated by trees
  - This is the only way of constructing structures in a Prolog
- Syntactically all abject in Prolog are "*terms*"

Some simple geometric objects



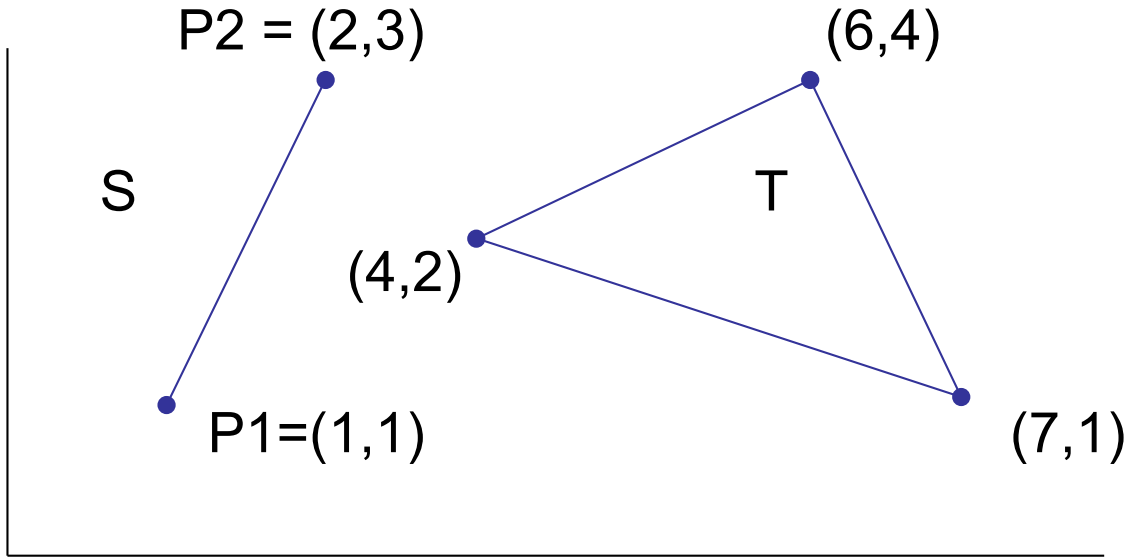
Points – point
Line segment – seg
Triangle – triangle

Some simple geometric objects

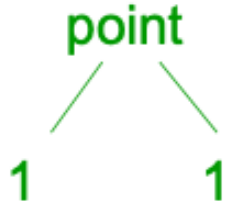


P1 = point(1, 1) P2 = point(2, 3)

Some simple geometric objects

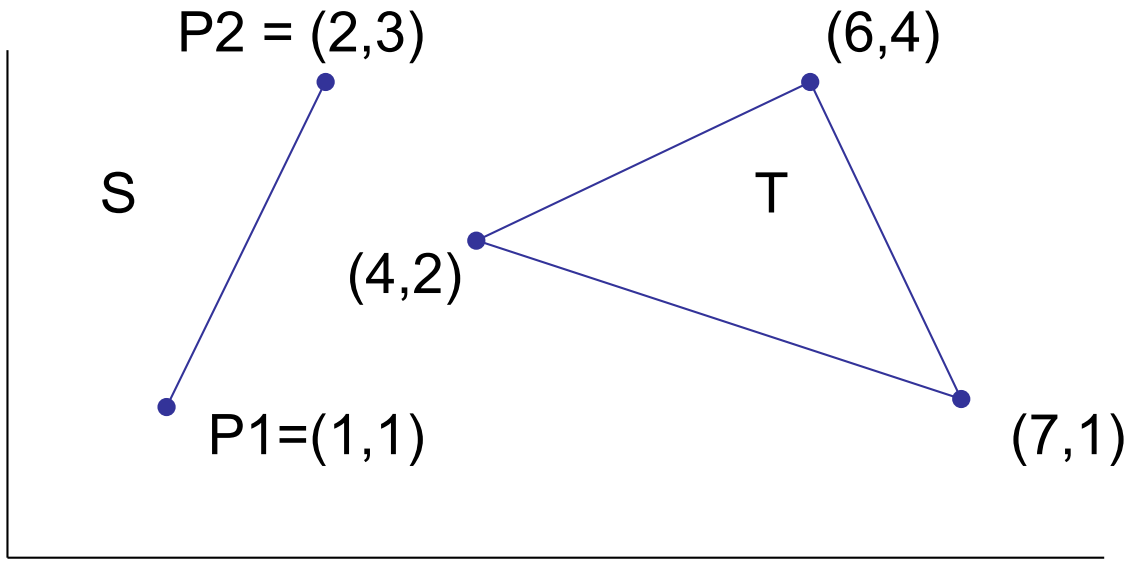


P1 = point(1, 1)



P2 = point(2, 3)

Some simple geometric objects

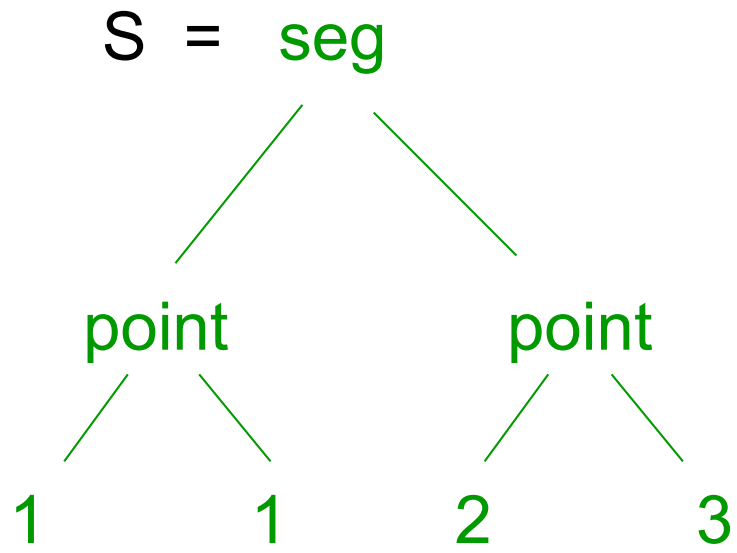


P1 = point(1, 1) P2 = point(2, 3)

S = seg(P1, P2) = seg(point(1,1), point(2,3))
T = triangle(point(4,2), point(6,4), point(7,1))

Segment

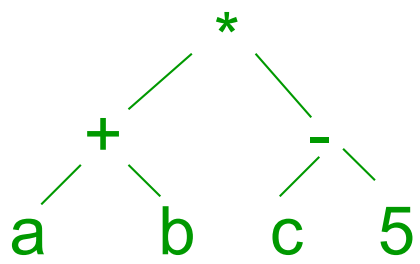
S = seg(point(1,1), point(2,3))

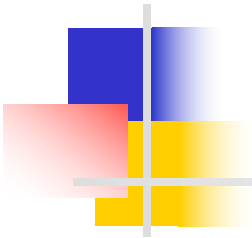


The Arithmetic Expressions are also Trees

- For example: (a + b) * (c - 5)
- Written as an expression with the functors:

*(+(a, b), -(c, 5))





Matching

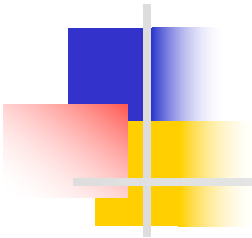
- Matching is an operation on *terms.*

Given two *terms, they match* if:

(1) They are identical, or

(2) The variable in both terms can be instantiated to objects in such a way that after the substitution of variables by these objects the terms become identical

- substitution of variable
  - the variable gets a value = instantiation of variable



Examples of *Matching*

- *Matching of dates*:

  date(D1, M1, 2006) = date(D2, june, Y2)

- One instantiation that make both trems identical

  D1 = D2

  M1 = june

  Y2 = 2006

- This is the most general instantiation, there are others that are less general…
- For matching using the operator "="

Matching- most general instantiation

- Prolog always returns the most general instantiation.
- With this instantiation leaves grater freedom for further instantiation if further *Matching* is required

?- date(D1, M1, 2006) = date(D2, june, Y2),
date(D1, M1, 2006) = date(17, M3, Y3).

D1 = 17, D2 = 17,
M1 = june, M3 = june,
Y2 = 2006, Y3 = 2006

Matching

- *Matching* succeeds or fails.

The genera rules:

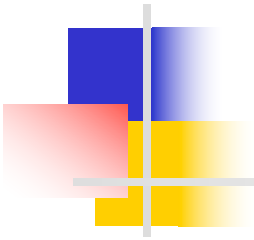
Two terms S and T match:

1. If S and T are constants, then they match only if they are identical
2. If S is a variable, and T is anything, the *Matching* succeeds, S becomes equal to T. Conversely , if T is a variable, then T is instantiated to S.
3. If S and T are structures then they match only if:

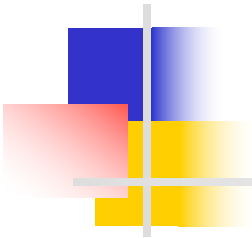
   a) they have the same principal functor and

   b) all their corresponding components match.

   The resulting instantiation is determined by the matching of the components.



Lists & Operatores



List

- The *list* is a special structure in Prolog.
- A list is a collection of terms, which is useful for grouping items together, or for dealing with large volumes of related data, etc.
- Examples :

```
[ a, b, c, d]   %Lists are enclosed by square brackets, and items are separated by commas.
                % The length of a list is the number of items it contains. The length of this list is 4.

[ ]             % Empry Lists
[ ann, tennis, tom, running]
[ link(a,b), link(a,c), link(b,d)]
[ a, [b,c], d, [ ], [a,a,a],   f(X,Y) ]
```

Representation of List with Head And Tail

- We can think of a list as being made up of two parts: the first element, known as the Head, and everything else, called the Tail.
- Prolog uses a built-in operator, the pipe (|) in order to give us this split for a list.

- The list

```
[red, white, black, yellow]
```

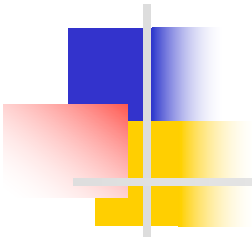
can be written as

```
[Head|Tail] = [red, white, black, yellow].
```

here is

Head = red

Tail = [white, black, yellow]



Representation of List with Head And Tail

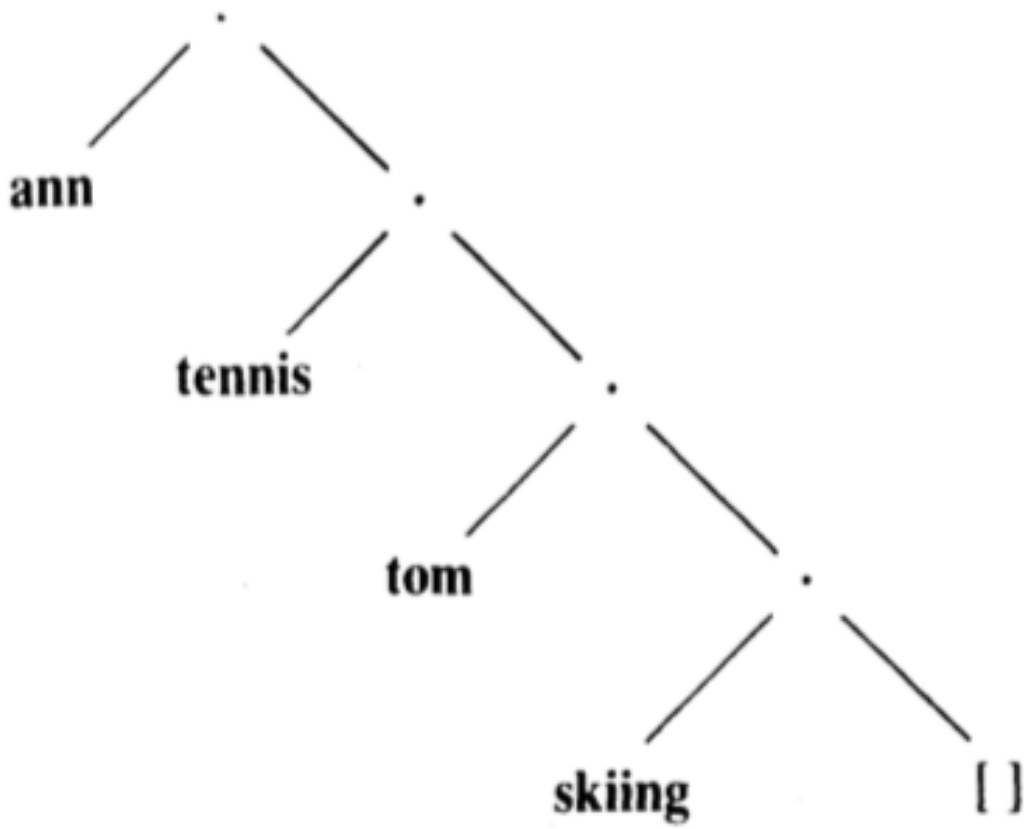
- Lists

L = [Head|Tail]
L = [a, b, c] = [a | [b, c]] = [a, b | [c]] = [a, b, c | [ ]]

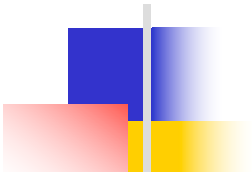
- In Prolog

.(Head, Tail)
[a,b,c] = .(a,.(b,.(c,[])))

Tree representation of a List




```
[ann, tennis, tom, skiing]
```



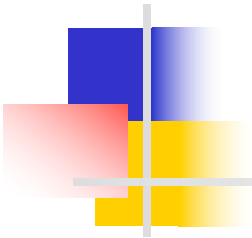
Representation of List

```
?- Hobbies1 = .( tennis, .( music, [] ) ),
   Hobbies2 = [ skiing, food],
   L = [ ann, Hobbies1, tom, Hobbies2].
```

```
Hobbies1 = [ tennis, music]
Hobbies2 = [ skiing, food]
L = [ ann, [tennis,music], tom, [skiing,food] ]
```

```
?- List1 = [a,b,c],
   List2 = .( a, .( b, .( c, [] ) ) ).
```

```
List1 = [a,b,c]
List2 = [a,b,c]
```



Membership – member/2

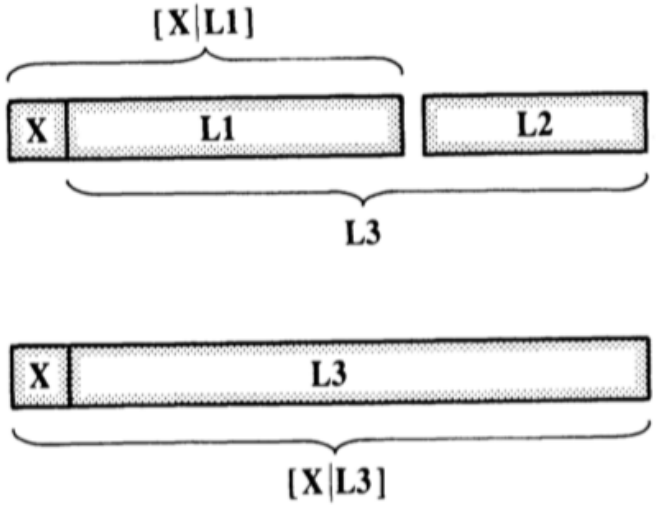
```
% member( X, L):  X is member of L

member( X, [ X | _]).      % X appears as head of list

member( X, [ _ | L]) :-
   member( X, L).          %  X in tail of list
```

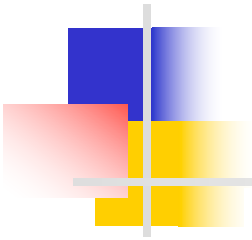
Different usage of member/2

```
?- member( c, [a,b,c,d]).        % Is an element of a given list
yes
?- member( X, [a,b,c,d]).        % Search for any element in the list
X = a;
X = b;
...
?- member( a, L).                % Find a List L containing "a"
L = [a | _ ] ;
L = [ _, a | _ ] ;               % "a" is the fist element  in L

L = [ _, _, a | _ ] ;            % "a" is the second element in  L
...
```

Concatenation Of Lists

```
% conc( L1, L2, L3): L3 is concatenation of L1 and L2

conc( [ ], L, L).                          % Base case

conc( [X | L1], L2, [X | L3]) :-           % Recursive case
    conc( L1, L2, L3).
```





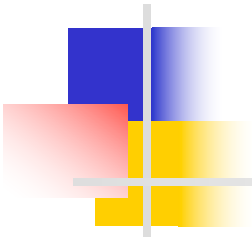
Examples of using Conc

?- conc([a,b,c], [1,2,3], L).
L = [a,b,c,1,2,3]

?- conc([a,[b,c],d], [a,[ ],b], L).
L = [a, [b,c], d, a, [ ], b]

?- conc(L1, L2, [a,b,c]).
L1 = [], L2 = [a,b,c];
L1 = [a], L2 = [b,c];
L1 = [a,b], L2 = [c];
L1 = [a,b,c], L2 = []

Example: Which months are before May, which after?



Try in Prolog

Write a Prolog Query:

Which months are before May, which after?

```
?- Months= [jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec] ,
   conc( Before, [may | After], Months).
```

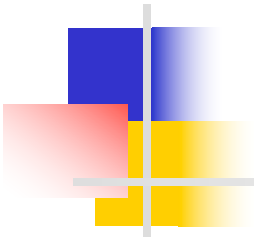
Example: Which months are before May, which after?

Try in Prolog

Write a Prolog Query:

Which months are before May, which after?

```
?- Months= [jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec] ,
    conc( Before, [may | After], Months).

   Before = [jan, feb,mar,apr],
   After = [jun,jul,aug,sep,oct,nov,dec]
```

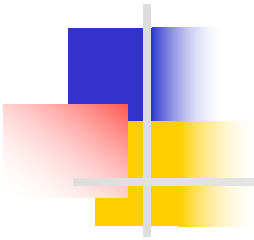


Exercise

Write a Prolog Query:

Delete from the list L1 everything from three 'z' onwards.

```
?- L1 = [a,b,z,z,c,z,z,z,d,e],        % A list L1 with a pattern
   conc(L2,[z,z,z|_],L1).            % L2 is L1 up to the pattern
```

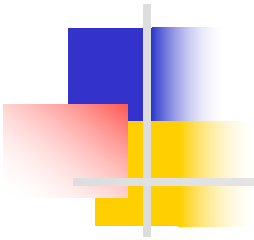


Exercise

Write a Prolog Query:

Delete from the list L1 everything from three 'z' onwards.

```
?- L1 = [a,b,z,z,c,z,z,z,d,e],        % A list L1 with a pattern
   conc(L2,[z,z,z|_],L1).            % L2 is L1 up to the pattern
```



Member by using Conc

- % member2(X, L): X is member of list L

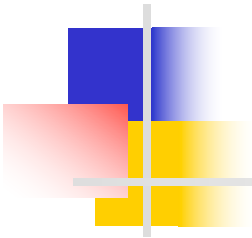
```
member2( X, L) :-
   conc( _, [X | _ ], L).
```

Deleting and inserting elements in a List

Delete element from a List
% del(X, L, NewL)

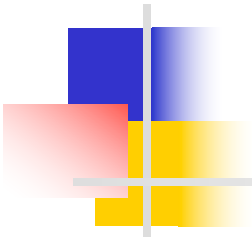
Insertion is the reverse operation of deletion
% insert(X, L, NewL)

You can also use the ”del" to insert elements in a list.



Sublist

```
% sublist( List, Sublist): Sublist appears as a sublist
% (subsection) in a List

sublist( S, L) :-
    conc( L1, L2, L),
    conc( S, L3, L2).
```

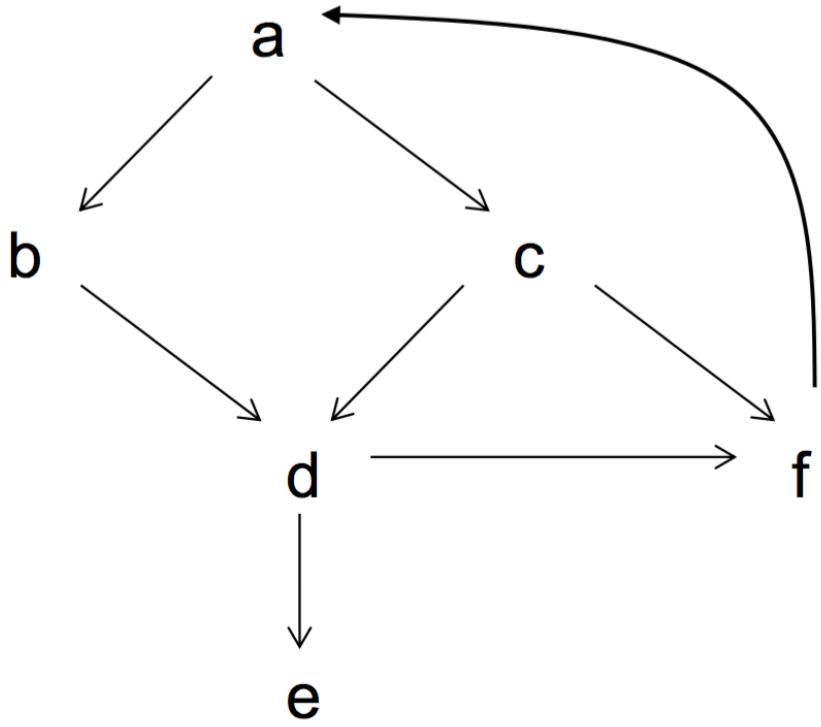


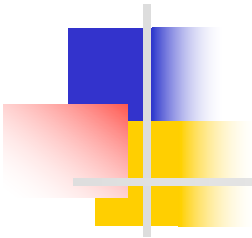
Definition of a List

```
list( [ ]).

list( [ _ | Tail]) :-
   list( Tail).

% Generate lists of incremental lengths
?- list( L).
L = [ ];
L = [ _A];
...
```

Path in a Graph

link(a, b).	**link(a, c).**
link(b, d).	**link(c, d).**
link(c, f).	**link(d, e).**
link(d, f).	**link(f, a).**

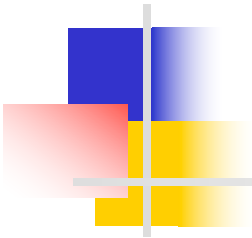




Path in a Graph

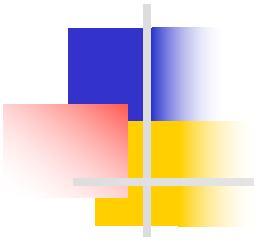
% path(StartNode, GoalNode): path exists between the nodes

```
path( Node, Node).

path( Start, End) :-
   link( Start, Next),
   path( Next, End).
```



Path in a Graph

```
% path( Start, Goal, Path):
% Path = list of nodes from Start to Goal

path( Start, Start, [Start]).

path( Start, Goal, [Start | Rest]) :-
   link( Start, Next),
   path( Next, Goal, Rest).
```

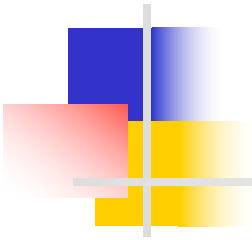


Exercise

Try:

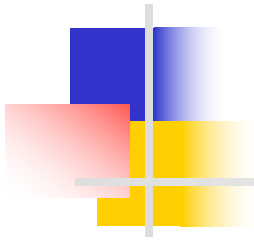
?- path(a, e, P).

?- path(a, c, P). % Problem: the search in depth misses the solution



Arithmetic in Prolog

Built-in Predicate for Arithmetic Operations



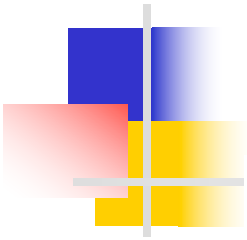
?- X = 1+2.

X=1+2

?- X is 1 + 2. % "is": built-in predicate that forces evaluation

X=3

A special predefined operator is to invoke arithmetic!



Arithmetic Operations

+, -, *, /, ** for real numbers
//, mod for integer
sin, cos, log, ... standard functions

?- X is 2 + sin(3.14/2).
X = 2.9999996829318345

?- A is 11/3.
A = 3.6666666666666665

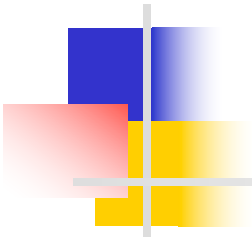
?- B is 11//3.
B=3

?-C is 11mod3.
C=2

Comparison operators

X > Y	X is greater than Y
X < Y	X is less than Y
X >= Y	X is greater than or equal to Y
X =< Y	X is less than or equal to Y
X =:= Y	the values of X and Y are equal
X =\= Y	the values of X and Y are not equal

```
?- 315*3 >= 250*4.
 no

?- 2+5 = 5+2.
no

?- 2+5 =:= 5+2.
yes
```



Comparison operators

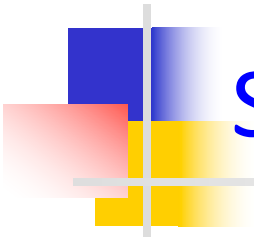
Operator

=:=

we have to distinguish from the operator

==

which serves to compare the non- arithmetic terms

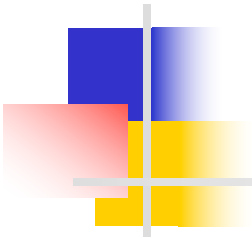


Strings

- Strings are lists of positive integers
  - Positive integers correspond to an ASCII code character.

Prolog does not make difference between

"Prolog” and [80,114,111,108,111,103]



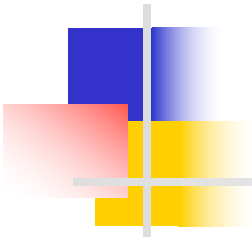
Strings

The procedure

`name(Atom,List)`

allows the conversion of the atom into the list of ASCII code characters

```
?- name(A, [112,114,111,108,111,103]).
 A = prolog          % No spaces without single quotation marks
```



- End