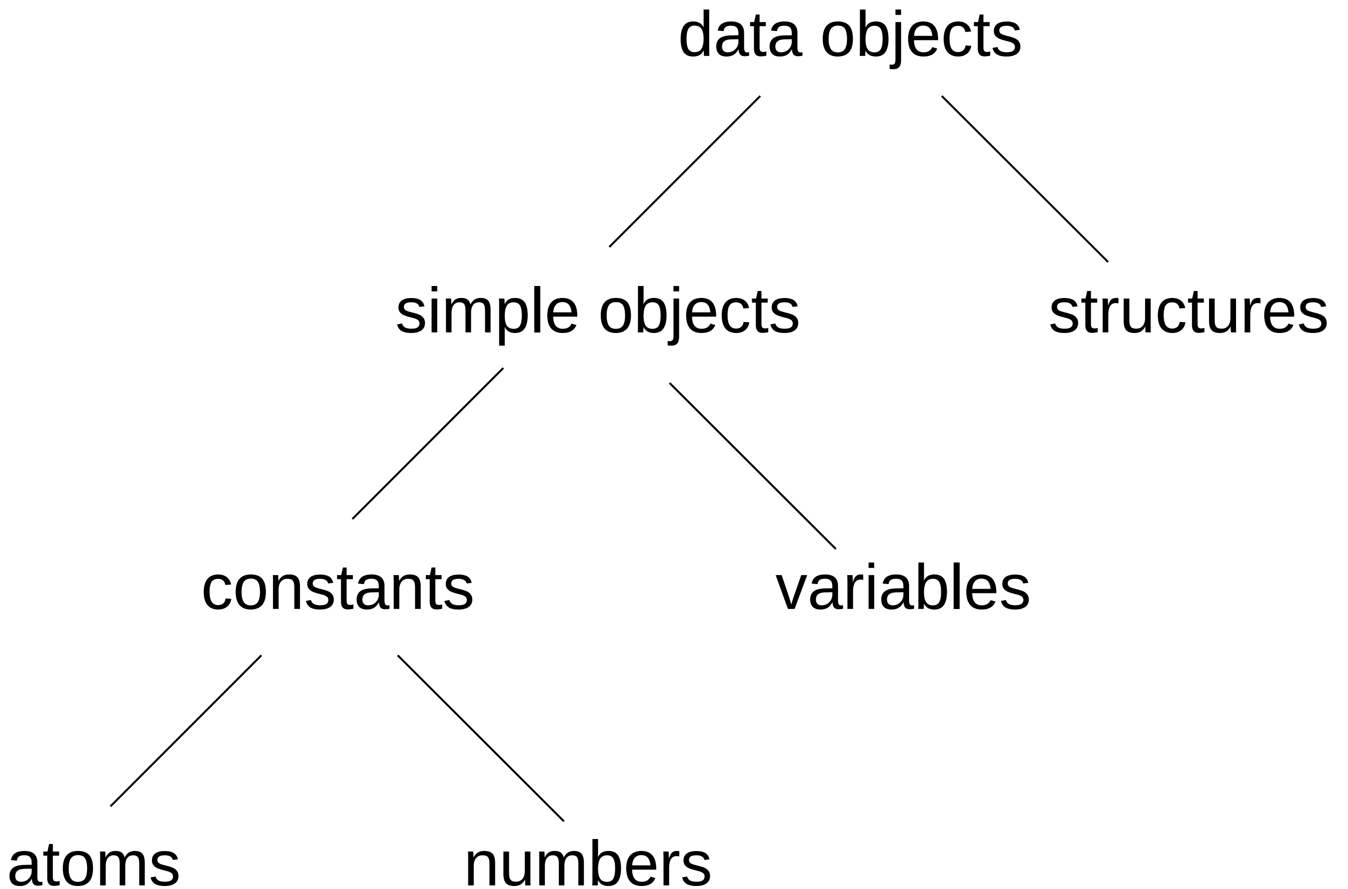
SYNTAX AND MEANING OF PROLOG PROGRAMS

Ivan Bratko

University of Ljubljana

These slides are meant to be used with a Prolog system to demonstrate the examples, and the book: I. Bratko, Prolog Programming for Artificial Intelligence, 4th edn., Pearson Education 2011. The slides alone are not self-sufficient.

DATA OBJECTS



SYNTAX FOR DATA OBJECTS

- Type of object always recognisable from its syntactic form

THREE SYNTACIC FORMS FOR ATOMS

(1) Strings of letters, digits, and “_”, starting with lowercase letter:

x x15 x_15 aBC_CBa7

alpha_beta_algorithm taxi_35

peter missJones miss_Jones2

ATOMS, CTD.

(2) Strings of special characters

    --->   <==>   <<

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

ATOMS, CTD.

(3) Strings in single quotes

‘X_35’ ‘Peter’ ‘Britney Spears’

SYNTAX FOR NUMBERS

- Integers

  1 1313 0 -55

- Real numbers (floating point)

  3.14 -0.0045 1.34E-21 1.34e-21

SYNTAX FOR VARIABLES

- Strings of letters, digits, and underscores, starting with uppercase letter

  X   Results   Object2B   Participant_list

  _x35   _335

- Lexical scope of variable names is one clause

- Underscore stands for an anonymous variable
- Each appearance of underscore: another anon. var.

ANONYMOUS VARIABLES

```
visible_block( B)  :-
  see( B, _, _).
```

Equivalent to:

```
visible_block( B)  :-
  see( B, X, Y).
```

STRUCTURES

- Structures are objects that have several components
- For example: dates are structured objects with three components
- Date 17 June 2006 can be represented by *term*:

date(17, june, 2006)

functor *arguments*

- An argument can be any object, also a structure

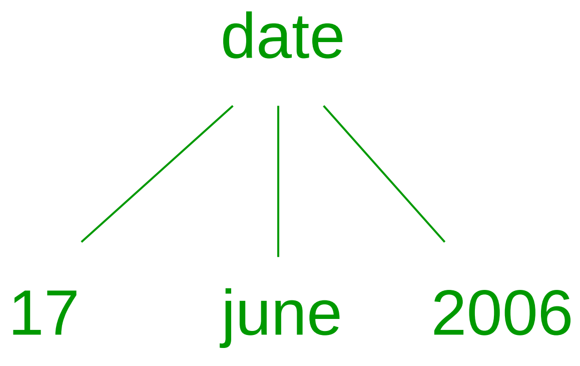
FUNCTORS

- Functor name chosen by user
- Syntax for functors: atoms
- Functor defined by name and *arity*

TREE REPRESENTATION OF STRUCTURES

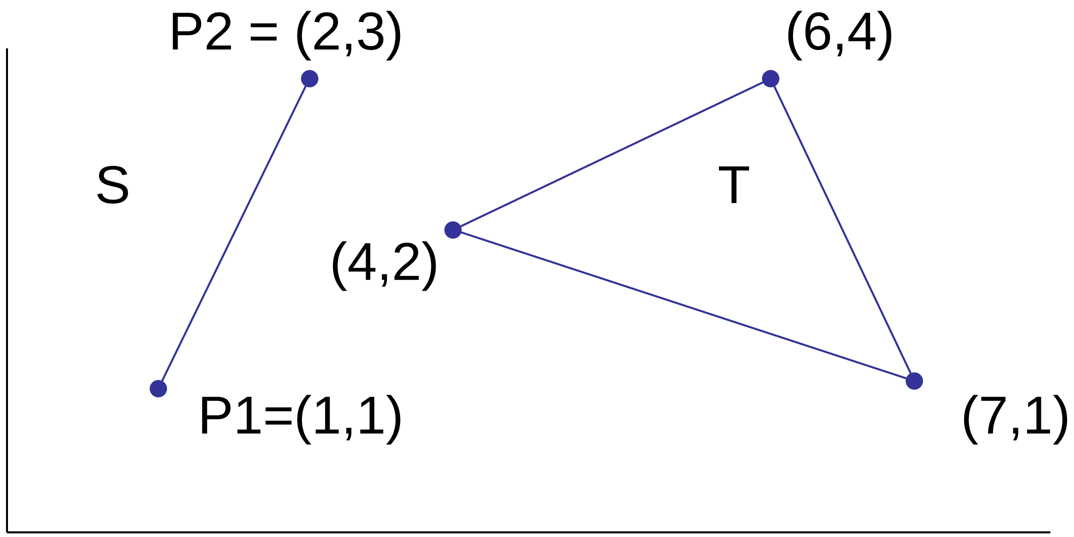
Often, structures are pictured as trees

date(17, june, 2006)



- Therefore all structured objects in Prolog can be viewed as trees
- This is the *only* way of building structured objects in Prolog

SOME GEOMETRIC OBJECTS



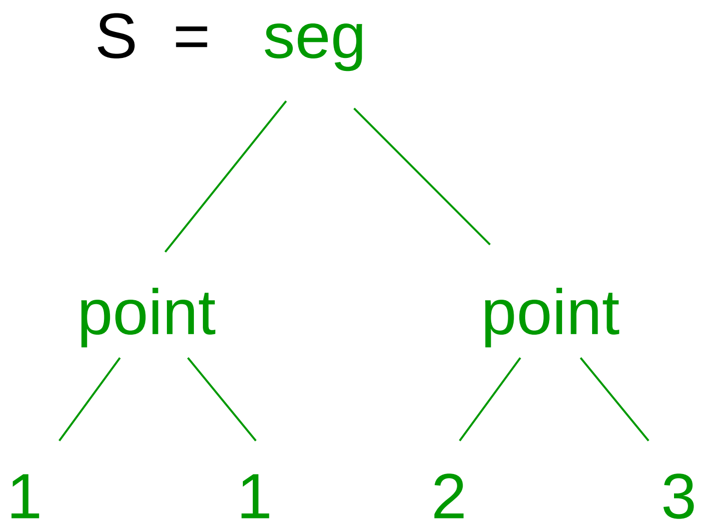
P1 = point(1, 1)　　　　　P2 = point(2, 3)

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

T = triangle(point(4,2), point(5,4), point(7,1))

LINE SEGMENT

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

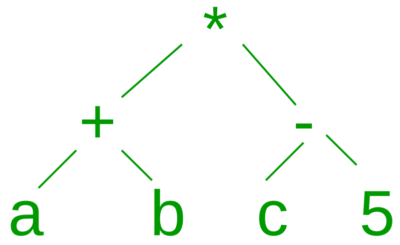


ARITHMETIC EXPRESSIONS ARE ALSO STRUCTURES

- For example: (a + b) * (c - 5)

- Written as term with functors:

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



MATCHING

- Matching is operation on terms (structures)
- Given two terms, they match if:

  (1) They are identical, or

  (2) They can be made identical by properly instantiating the variables in both terms

EXAMPLE OF MATCHING

- Matching two dates:

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

- This causes the variables to be instantianted as:

  D1 = D2

  M1 = june

  Y2 = 2006
- This is the *most general instantiation*
- A less general instantiation would be: D1=D2=17, ...

MOST GENERAL INSTANTIATION

- In Prolog, matching always results in most general instantiation
- This commits the variables to the least possible extent, leaving flexibility for further instantiation if required
- For example:

```
?- 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; if succeeds then it results in the most general instantiation

- To decide whether 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 then matching succeeds, S is instantiated to T; analogously if T is a variable

  (3) If S and T are structures then they match only if

    (a) they both have the same principal functor, and

    (b) all their corresponding arguments match

MATCHING ≈ UNIFICATION

- Unification known in predicate logic
- Unification = Matching + Occurs check
- What happens when we ask Prolog:

```
?-  X = f(X).
```

Matching succeeds, unification fails

COMPUTATION WITH MATCHING

```
% Definition of vertical and horizontal segments

vertical( seg( point( X1,Y1), point( X1, Y2))).
horizontal( seg( point( X1,Y1), point( X2, Y1))).

?- vertical( seg( point( 1,1), point( 1, 3))).
yes
?- vertical( seg( point( 1,1), point( 2, Y))).
no
?- vertical( seg( point( 2,3), P)).
P = point( 2, _173).
```

AN INTERESTING SEGMENT

- Is there a segment that is both vertical and horizontal?

```
?-  vertical( S), horizontal( S).

S = seg( point( X,Y), point(X,Y))
```

- Note, Prolog may display this with new variables names as for example:

```
S = seg( point( _13,_14), point( _13, _14))
```

DECLARATIVE MEANING

- Given a program P and a goal G,

  G is true (i.e. logically follows from P) if and only if:

  (1) There is a clause C in P such that

  (2) there is a clause instance I of C such that

  (a) the head of I is identical to G, and

  (b) all the goals in the body of I are true

- An *instance* of a clause C is obtained by renaming each variable in C and possibly substituting the variable by some term. E.g. an instance of

  p(X,Y) :- q(Y,Z)

  is

  p(U,a) :- q(a,V).

DECLARATIVE vs PROCEDURAL MEANING OF PROLOG PROGRAMS

- Consider:

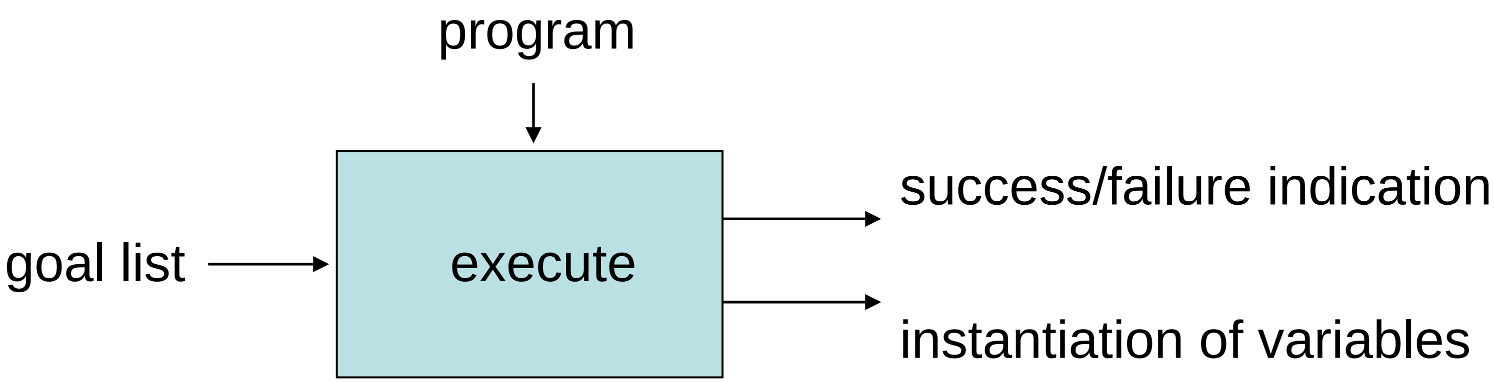
```
P :- Q, R.
```

- Declarative readings of this:
  - P is true if Q *and* R are true.
  - From Q *and* R follows P.

- Procedural readings:
  - To solve problem P, *first* solve subproblem Q and *then* R.
  - To satisfy P, *first* satisfy Q and *then* R.

PROCEDURAL MEANING

- Specifies how Prolog answers questions
- Procedural meaning is an algorithm to execute a list of goals given a Prolog program:



procedure execute(Program, GoalList, Success)

- execute = declarative meaning + procedural elements

Search program from top to bottom to find such clause

G is true (i.e. logically follows from P) if and only if:

(1) there is a clause C in P such that

(2) there is a clause instance I of C such that

(a) the head of I is identical to G, and

(b) all the goals in the body of I are true

Match G and head of C

Execute goals in order as they appear in program