Prologue

- Stands for *PROgramming in LOGic*

- Basic idea of logic programming:

  - describe the solution to a problem using logical statements

  - do not specify how to solve the problem; the system will perform logical deduction on the statements to find a solution

  - “declarative”, not “procedural”, programming

- Practical Prolog implementations provide some procedural features

A Short History of Prolog

- 1970: The first implementations of logic programming languages were written in LISP
  - Planner, Microplanner
- 1975: Efficient implementation of Prolog by D. Warren at Edinburgh
- 1980: Prolog selected for Japan's Fifth Generation project
- 1990: Prolog is the dominant AI language in Europe and Japan, second to LISP in U.S.

Prolog programs

- ✔ Prolog is suited to applications in symbolic computing that involve objects and relations between objects
- ✔ For example, you can represent the fact that Tom is a parent of Bob as the Prolog clause

```
parent(tom,bob).
```

- ✔ Consider representing this family tree as a set of Prolog clauses...

Fig 1.1
Bratko

Prolog programs

- Put the following Prolog clauses in a file named **`defs`**:

```
parent(tom,bob).
parent(pam,bob).
parent(tom,liz).
parent(bob,ann).
parent(bob,pat).
parent(pat,jim).
```

- This is a Prolog program...!

- Start Prolog:

```
%prolog
SB-Prolog Version 3.1
| ?-
```

- Load the program:

```
| ?- consult(defs).
yes
| ?-
```

- Now you're ready to "run" the program...

Prolog programs

- ✔ You interact with the Prolog program by giving it goals to satisfy

- ✔ These goals are like questions to answer: Is this logical expression deducible from known facts and rules, and if so, for what value of the variables? For example:

  Is Bob a parent of Pat?

```
| ?- parent(bob,pat).
yes
```

  Is Liz a parent of Pat?

```
| ?- parent(liz,pat).
no
```

  Is Abraham a parent of Isaac?

```
| ?- parent(abraham,isaac)
no
```

Prolog programs

✔ Asing more questions...

Who is a parent of Liz?

```
| ?- parent(X,liz).
X = tom
yes
```

Who has Bob as parent?

```
| ?- parent(bob,Z).
Z = ann ;
Z = pat ;
no
```

Who is related by the parent relation?

```
| ?- parent(Parent,Child).
Parent = pam
Child = bob ;
Parent = tom
Child = bob ;
Parent = tom
Child = liz ;
Parent = bob
Child = ann
yes
```

Prolog programs

- ✔ Asking more questions...

Who is a grandparent of Jim?

If X is a grandparent of Jim, there is some Y such that X is parent of Y, and Y is parent of Jim

```
?- parent(X,Y), parent(Y,jim).
X = bob
Y = pat
yes
```

Rules in Prolog programs

- Besides fact clauses like

```
parent(pam,bob).
```

which can be read as

"Pam is a parent of Bob",

Prolog programs can contain rule clauses like

```
offspring(X,Y) :- parent(Y,X).
```

which can be read as

For all X and Y, if Y is a parent of X, then X is an offspring of Y.

$$\forall x \forall y\, (parent\,(y, x) \rightarrow offspring\,(x, y)\,)$$

Here

`offspring(X,Y)` is the *head* or *consequent* or *conclusion* of the rule
`parent(Y,X)` is the *body* or *antecedent* or *condition* of the rule

Rules in Prolog programs

- ✔ The scope of variables extends over the clause
- ✔ The body of the rule may be conjunctive
- ✔ Define the grandparent relation:

  For all X and Y, X is a grandparent of Y if there exists a Z such that X is parent of Z and Z is parent of Y

$$\forall x \forall y\, (\exists z\, (parent\,(x, z) \land parent\,(z, y)\,) \rightarrow grandparent\,(x, y)\,)$$

```
grandparent(X,Y) :- parent(X,Z), parent(Z,Y).
```

Rules in Prolog programs

- ✔ Rules in Prolog may be recursive
- ✔ Define the ancestor relation:

For all X and Y, X is an ancestor of Y if X is a parent of Y, or if there exists a Z such that Z is a parent of Y and X is an ancestor of Z

$$\forall x \forall y\, (parent\,(x, y) \rightarrow ancestor\,(x, y))$$

$$\forall x \forall y\, (\exists z\, (parent\,(z, y) \wedge ancestor\,(x, z)) \rightarrow ancestor\,(x, y))$$

```
ancestor(X,Y) :- parent(X,Y).

ancestor(X,Y) :- parent(Z,Y), ancestor(X,Z).
```

Rules in Prolog programs

- ✔ With these rules loaded into Prolog, we can ask these questions:

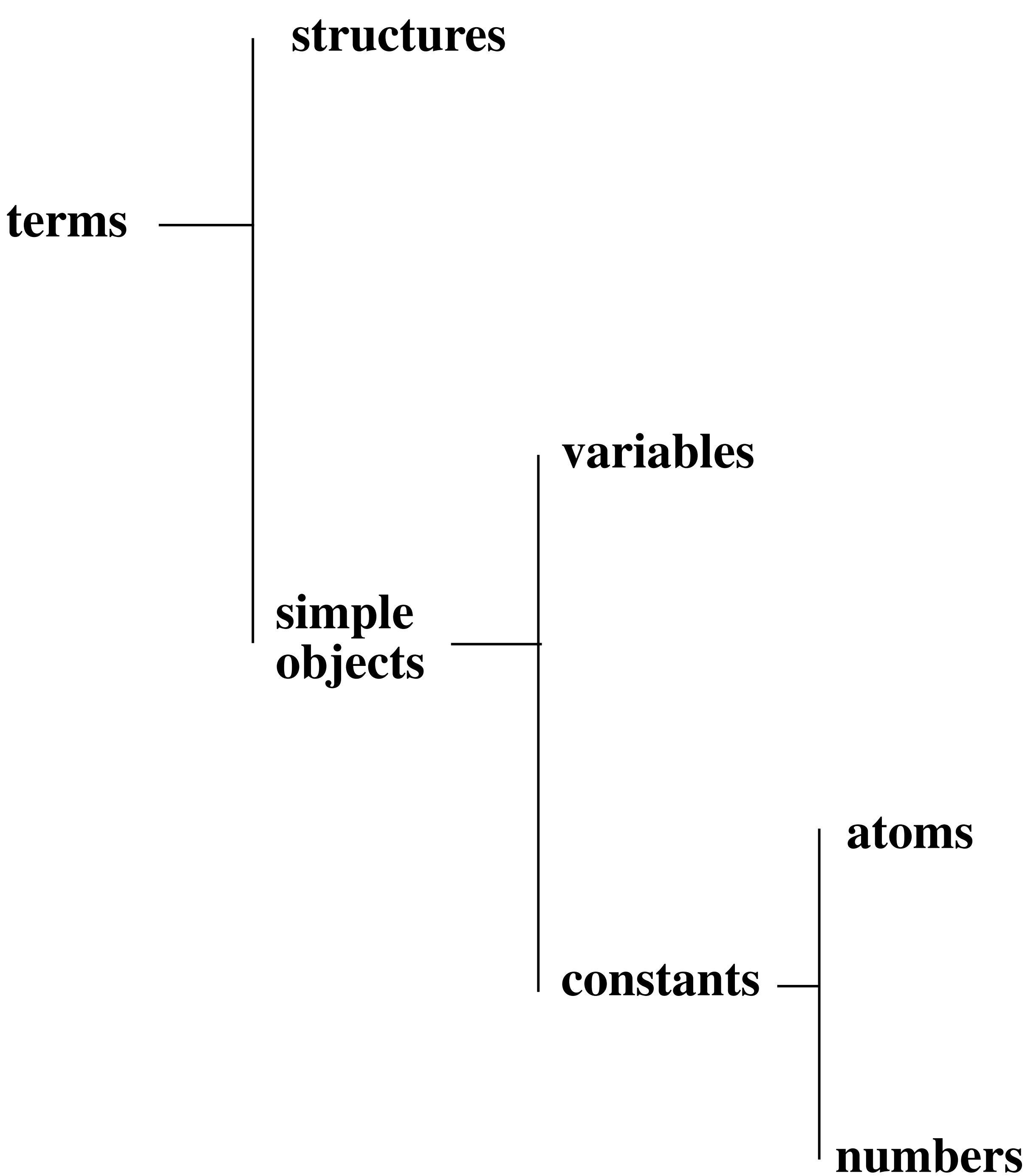
```
?- parent(pat,jim).
yes
```

```
?- offspring(jim,pat).
yes
```

```
?- grandparent(bob,jim).
yes
```

```
?- ancestor(X,jim).
X = pat;
X = bob;
X = pam;
X = tom;
no
```

Expression types in Prolog



Expressions in Prolog

- Variables:
  - expressed as strings of letters, digits, and underscrore characters
  - first character must be an upper-case letter or underscore
  - a single underscore '_' is an *anonymous variable*

```
X
Variable
The_first_name
Obj86
_zzyzx
_
```

Expressions in Prolog

- ✔ Numbers:
  - ✗ expressed in the usual notation for decimal integers
  - ✗ most implementations also have floating point numbers
  - ✗ built in support for numerical computation is not extensive

```
3.14159

-1024

98.6
```

Expressions in Prolog

- ✔ Atoms:
  - ✗ expressed as strings of letters, digits, and underscore characters
  - ✗ first character must be a lower-case letter
  - ✗ or: any string of characters enclosed in single quotes

```
foobar

fooBar

jane_Doe

plan9

'Jane_Doe'

'Jane Doe'

'** This is an ATOM??? )^%@#**'
```

Expressions in Prolog

- Structures:
  - a structure (or compound term) consists of a *functor* and a sequence of *arguments*
    - the functor must be an atom
    - the arguments may be any terms

```
greater-than(2,1)

greater-than(2,X)

greater-than(8,Y,5)

sentence(np(john),vp(v(likes),np(mary)))
```

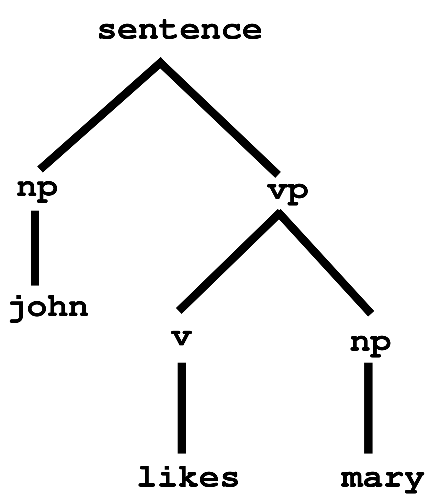
  - the usual form for structures is
    functor (arg1 , arg2 , ... , argn)
  - however, prefix and infix functors exist and can be defined...

    `3 < 4` is equivalent to `<(3,4)`
    `3 == 4` is equivalent to `==(3,4)`
    `X * 3.14` is equivalent to `*(X,3.14)`
    `f(bar,baz) = f(X,Y)` is equivalent to `=(f(bar,baz),f(X,Y))`
    `[a,b,c,d]` is equivalent to `.(a,.(b,.(c,.(d,[]))))`

Structures in Prolog

- ✔ Structures can be thought of as trees
  - ✗ the principal functor is at the root
  - ✗ arguments of the principlal functor (which may be structures) are children of the root

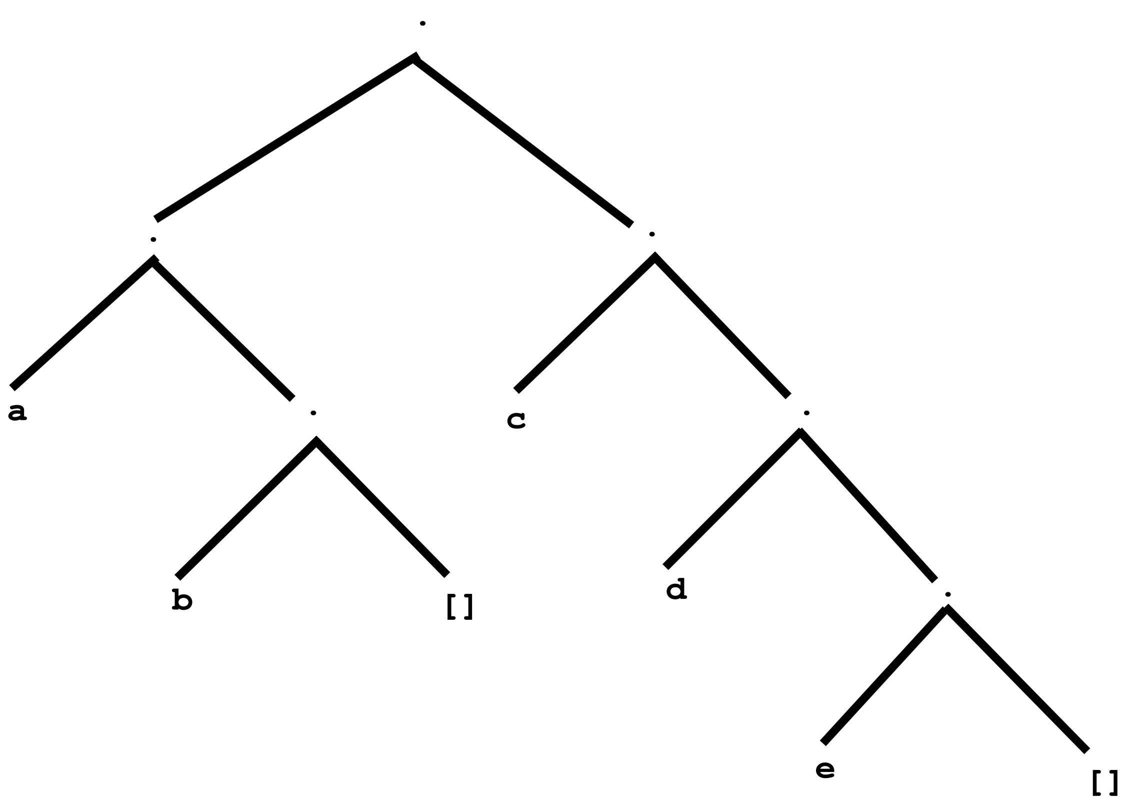
```
sentence(np(john),vp(v(likes),np(mary)))
```



Structures in Prolog

- ✔ lists in Prolog are just another kind of structure
- ✔ "." is a functor that acts like `cons` in LISP
- ✔ `[]` is the empty list

```
.(.(a,.(b,[])),.(c,.(d,.(e,[]))))
```



Matching in Prolog

- ✔ Matching (also called *unification*) of terms is the most important operation in the execution of Prolog programs

- ✔ Two terms S, T match if:
  - ✗ S and T are constants and they are identical, or
  - ✗ S is a variable, and T is anything; then S is instantiated to T, or
  - ✗ T is a variable, and S is anything; then T is instantiated to S, or
  - ✗ S and T are structures, and
    - S and T have the same principal functor, and
    - all their arguments match

Matching in Prolog

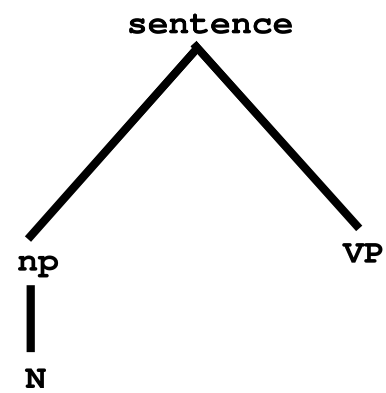
✔ Examples, expressed using the infix functor `=`:

```
foo = foo

foo = X

parent(mary,mike) = X

parent(mary,Y) = parent(Z,bob)

.(a,.(b,.(c,[]))) = .(First,Rest)

sentence(np(john),VP) = sentence(np(N),vp(V,np(mary)))
```

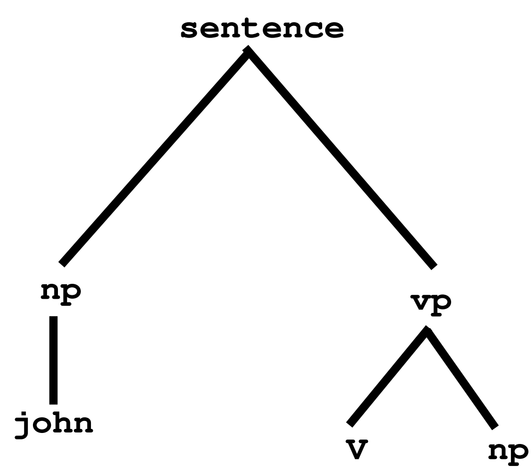
Matching in Prolog

- When thinking about matching complicated structures, it helps to think about their tree representations

`sentence(np(N),VP)`



`sentence(np(john),vp(V,np(mary)))`



these structures match, with instantiations

`N = john, VP = vp(V,np(mary)), V = V`