Lecture 3: Recursion

- Introduce recursive definitions in Prolog
- Show that there can be mismatches between the declarative and procedural meaning of a Prolog program

Recursive Definitions

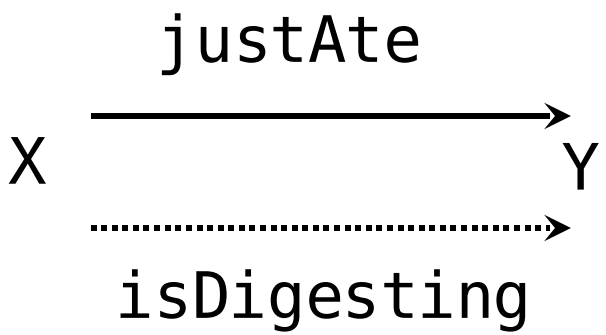
- Prolog predicates can be defined recursively
- A predicate is recursively defined if one or more rules in its definition refers to itself

Example 1: Eating

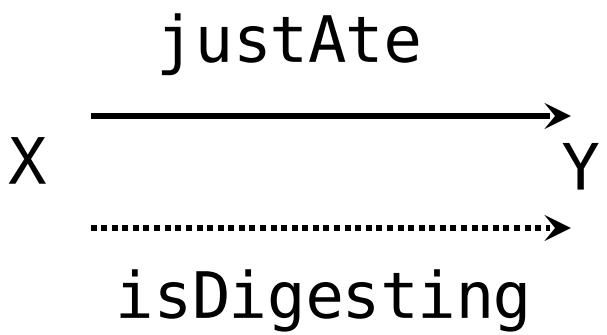
```
isDigesting(X,Y):- justAte(X,Y).
isDigesting(X,Y):- justAte(X,Z), isDigesting(Z,Y).

justAte(mosquito,blood(john)).
justAte(frog,mosquito).
justAte(stork,frog).
```

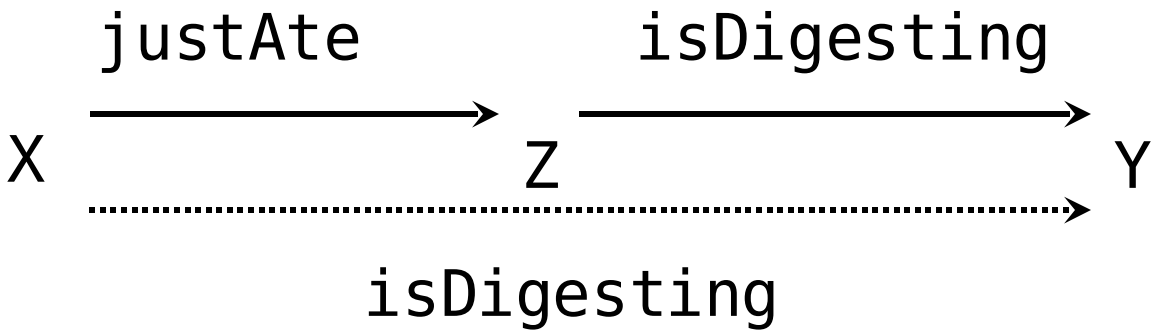
```
?-
```

Picture of the situation



Picture of the situation





Example 1: Eating

```
isDigesting(X,Y):- justAte(X,Y).
isDigesting(X,Y):- justAte(X,Z), isDigesting(Z,Y).

justAte(mosquito,blood(john)).
justAte(frog,mosquito).
justAte(stork,frog).
```

```
?- isDigesting(stork,mosquito).
```

Another recursive definition

```
p:- p.
```

```
?-
```

Another recursive definition

```
p:- p.
```

```
?- p.
```

Another recursive definition

```
p:- p.
```

```
?- p.
ERROR: out of memory
```

Example 2: Decendant

```
child(bridget,caroline).
child(caroline,donna).

descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), child(Z,Y).
```

Example 2: Decendant

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).


descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), child(Z,Y).
```

Example 2: Decendant

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).


descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), child(Z,Y).
```

```
?- descend(anna,donna).
no
?-
```

Example 2: Decendant

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).


descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), child(Z,Y).
descend(X,Y):- child(X,Z), child(Z,U), child(U,Y).
```

```
?-
```

Example 2: Decendant

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).


descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), descend(Z,Y).
```

```
?-
```

Example 2: Decendant

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).


descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), descend(Z,Y).
```

```
?- descend(anna,donna).
```

Example 3: Successor

- Suppose we use the following way to write numerals:
  1. **0** is a numeral.
  2. If **X** is a numeral, then so is **succ(X)**.

Example 3: Successor

```
numeral(0).
numeral(succ(X)):- numeral(X).
```

Example 3: Successor

```
numeral(0).
numeral(succ(X)):- numeral(X).
```

```
?- numeral(succ(succ(succ(0)))).
yes
?-
```

Example 3: Successor

```
numeral(0).
numeral(succ(X)):- numeral(X).
```

```
?- numeral(X).
```

Example 3: Successor

```
numeral(0).
numeral(succ(X)):- numeral(X).
```

```
?- numeral(X).
X=0;
X=succ(0);
X=succ(succ(0));
X=succ(succ(succ(0)));
X=succ(succ(succ(succ(0))))
```

Example 4: Addition

?- add(succ(succ(0)),succ(succ(succ(0))), Result).

Result=succ(succ(succ(succ(succ(0)))))

yes

Example 4: Addition

```
add(0,X,X).                    %%% base clause
```

```
?- add(succ(succ(0)),succ(succ(succ(0))), Result).
Result=succ(succ(succ(succ(succ(0)))))
yes
```

Example 4: Addition

```
add(0,X,X).                          %%% base clause

add(succ(X),Y,succ(Z)):-            %%% recursive clause
      add(X,Y,Z).
```

```
?- add(succ(succ(0)),succ(succ(succ(0))), Result).
Result=succ(succ(succ(succ(succ(0)))))
yes
```

Prolog and Logic

- Prolog was the first reasonable attempt to create a logic programming language
  - Programmer gives a declarative specification of the problem, using the language of logic
  - The programmer should not have to tell the computer what to do
  - To get information, the programmer simply asks a query

Prolog and Logic

- Prolog does some important steps in this direction, but nevertheless, Prolog is not a full logic programming language!
- Prolog has a specific way of answering queries:
  - Search knowledge base from top to bottom
  - Processes clauses from left to right
  - Backtracking to recover from bad choices

descend1.pl

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).

descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), descend(Z,Y).
```

```
?- descend(A,B).
A=anna
B=bridget
```

descend2.pl

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).

descend(X,Y):- child(X,Z), descend(Z,Y).
descend(X,Y):- child(X,Y).
```

```
?- descend(A,B).
A=anna
B=emily
```

descend3.pl

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).

descend(X,Y):- descend(Z,Y), child(X,Z).
descend(X,Y):- child(X,Y).
```

```
?- descend(A,B).
ERROR: OUT OF LOCAL STACK
```

descend4.pl

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).

descend(X,Y):- child(X,Y).
descend(X,Y):- descend(Z,Y), child(X,Z).
```

```
?- descend(A,B).
```

Summary of this lecture

- In this lecture we introduced recursive predicates
- We also looked at the differences between the declarative and the procedural meaning of Prolog programs
- We have identified some of the shortcomings of Prolog seen as a logical programming language

Exercises (1)

Define a predicate greater_than/2 that takes two numerals in the notation that we introduced in this lecture (i.e. 0, succ(0), succ(succ(0)) ...) as arguments and decides whether the first one is greater than the second one. E.g:

```
?- greater_than(succ(succ(succ(0))),succ(0)).
yes
?- greater_than(succ(succ(0)),succ(succ(succ(0)))).
no
```

Exercises (2)

Binary trees are trees where all internal nodes have exactly two children. The smalles binary trees consist of only one leaf node. We will represent leaf nodes as leaf(Label). For instance, leaf(3) and leaf(7) are leaf nodes, and therefore small binary trees.
Given two binary trees B1 and B2 we can combine them into one binary tree using the predicate tree: tree(B1,B2). So, from the leaves leaf(1) and leaf(2) we can build the binary tree tree(leaf(1), leaf(2)). And from the binary trees tree(leaf(1), leaf(2)) and leaf(4) we can build the binary tree tree(tree(leaf(1), leaf(2)), leaf(4)).

Now, define a predicate swap/2, which produces a mirror image of the binary tree that is its first argument. For example:

```
?- swap(tree(tree(leaf(1), leaf(2)), leaf(4)),T).
T = tree(leaf(4), tree(leaf(2), leaf(1))).
yes
```

Exercises (3)

We have the following knowledge base:

```
directTrain(forbach,saarbruecken).
directTrain(freyming,forbach).
directTrain(fahlquemont,stAvold).
directTrain(stAvold,forbach).
directTrain(saarbruecken,dudweiler).
directTrain(metz,fahlquemont).
directTrain(nancy,metz).
```

- Write a recursive predicate travelBetween/2 that tells us when we can travel by train between two towns.

- It is, furthermore, plausible to assume that whenever it is possible to take a direct train from A to B, it is also possible to take a direct train from B to A. Can you encode this in Prolog?

  You program should e.g. answer `yes' to the following query:
  travelBetween(saarbruecken,nancy).

- Do you see any problems you program may run into?