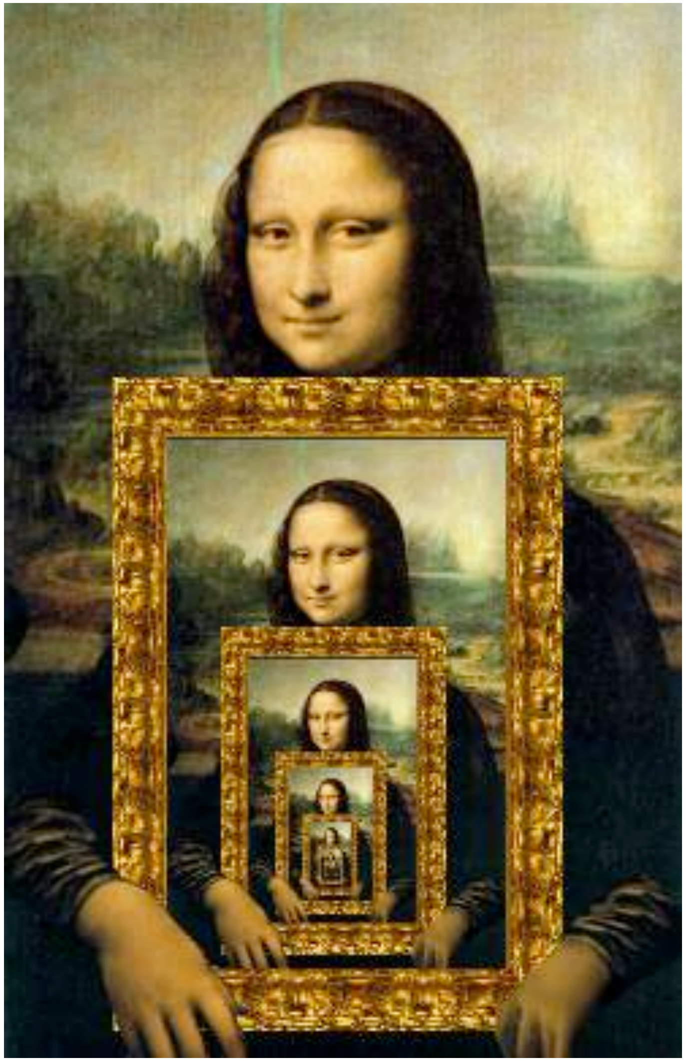
Chapter 3: Recursion

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

- Exercises
  - Exercises of LPN chapter 3
  - Practical work

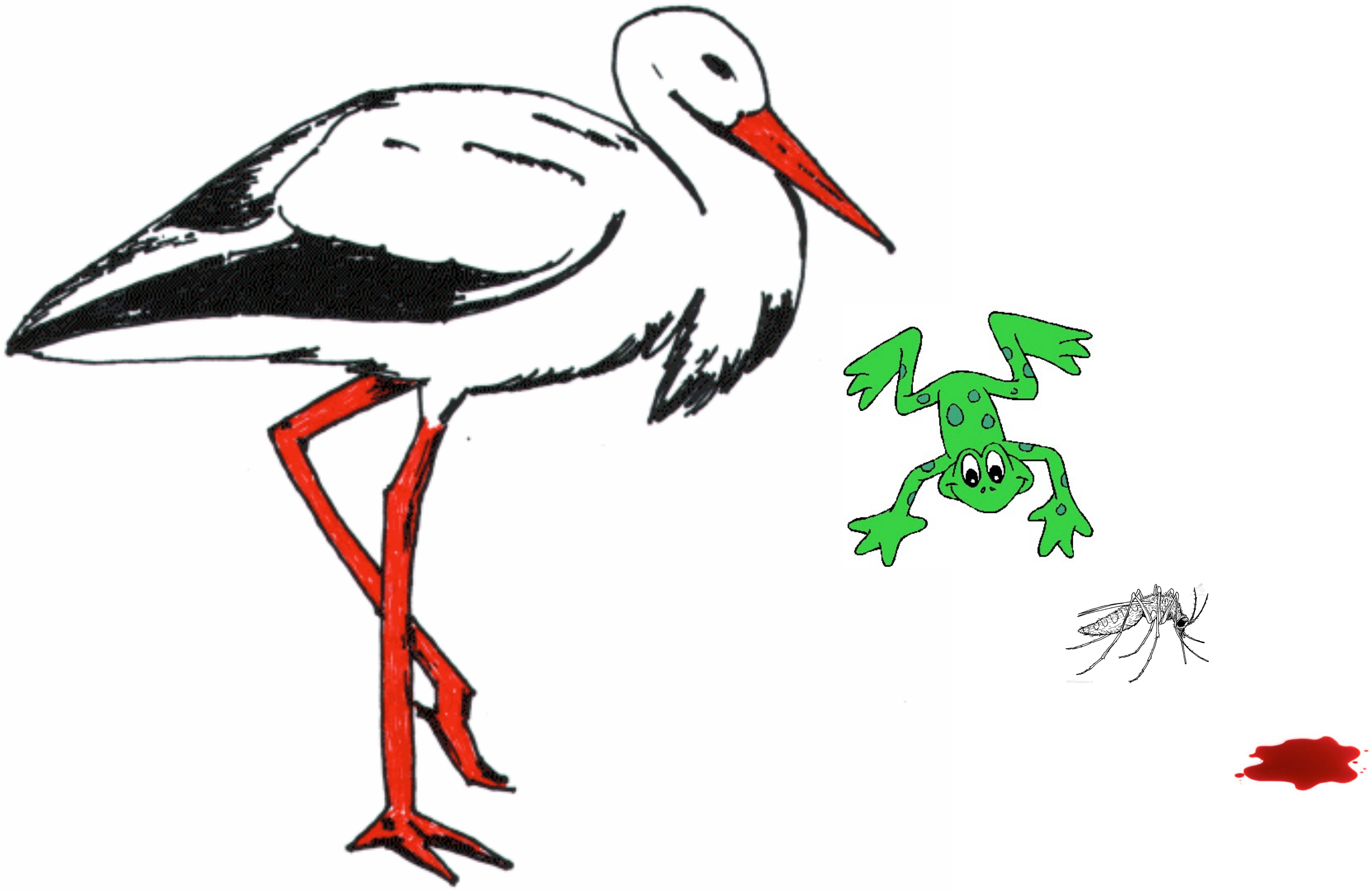
Chapter 3: Recursion



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

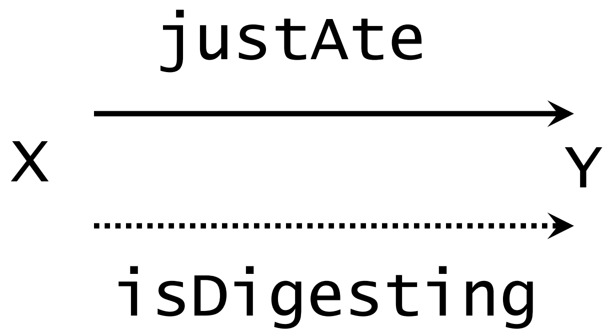


Example 1: Eating

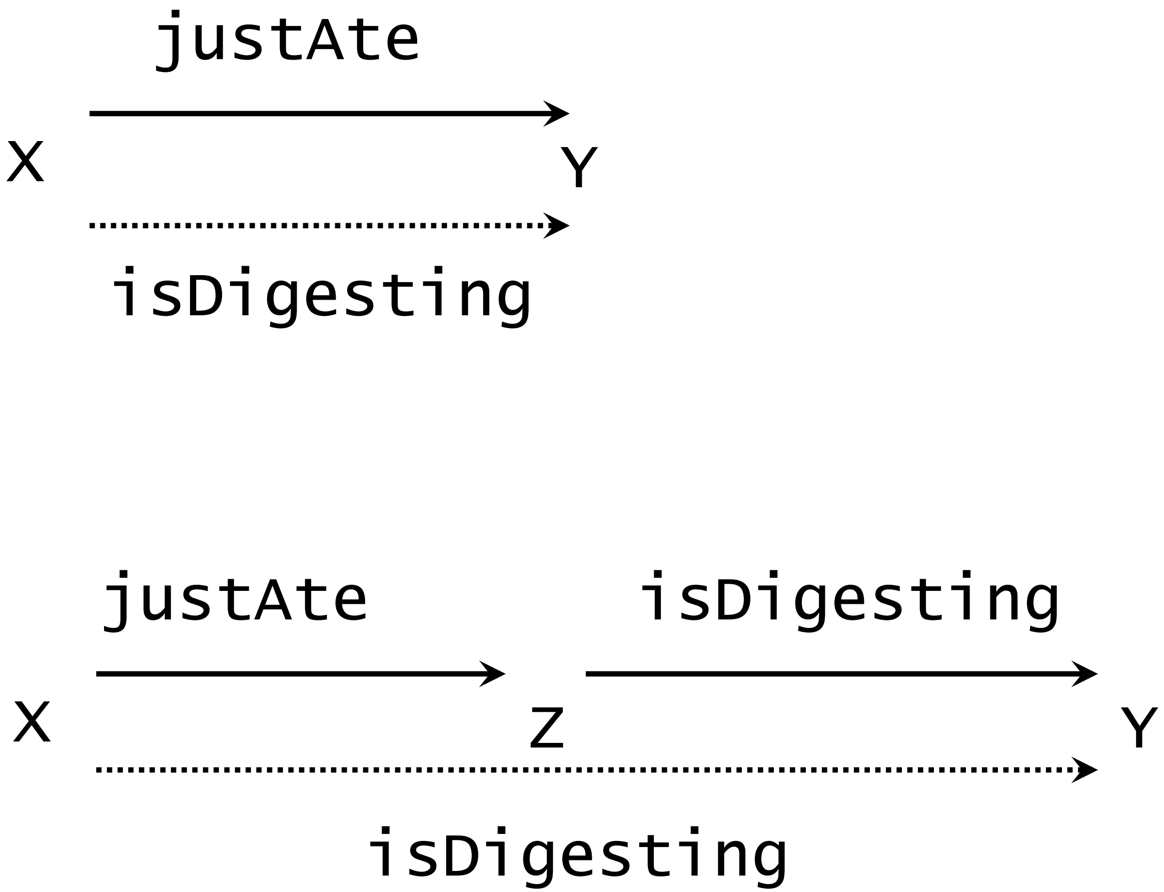
```
isDigesting(X,Y):- justAte(X,Y).
isDigesting(X,Y):- justAte(X,Z), isDigesting(Z,Y).

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).
```

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).
yes
?-
```

Another recursive definition

```
p:- p.
```

```
?-
```

Another recursive definition

```
p:- p.
```

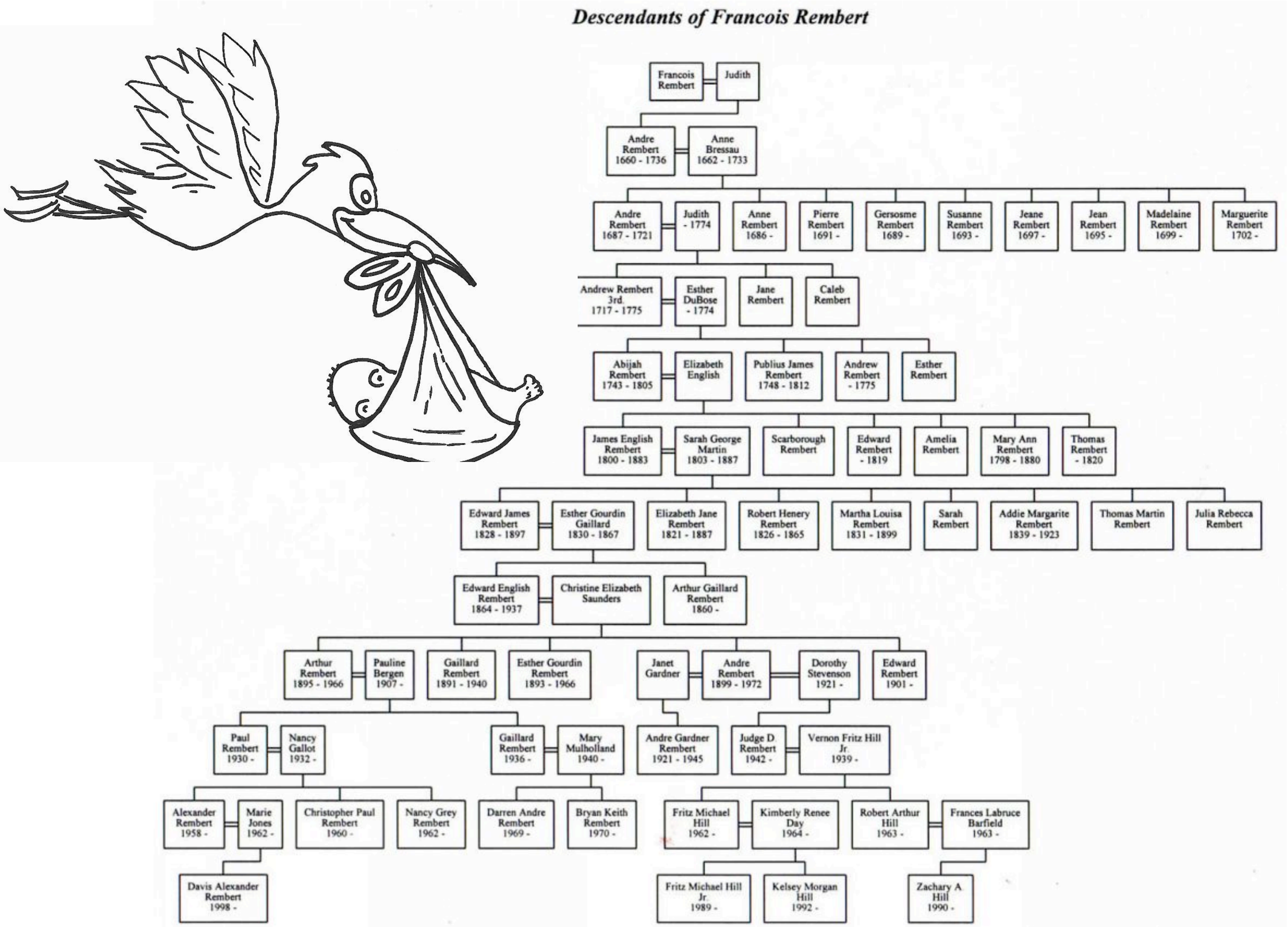
```
?- p.
```

Another recursive definition

```
p:- p.
```

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

Example 2: Decendant



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).
```

Search tree

Draw search tree for

?- descend(anna,donna).

Example 3: Successor

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).
```

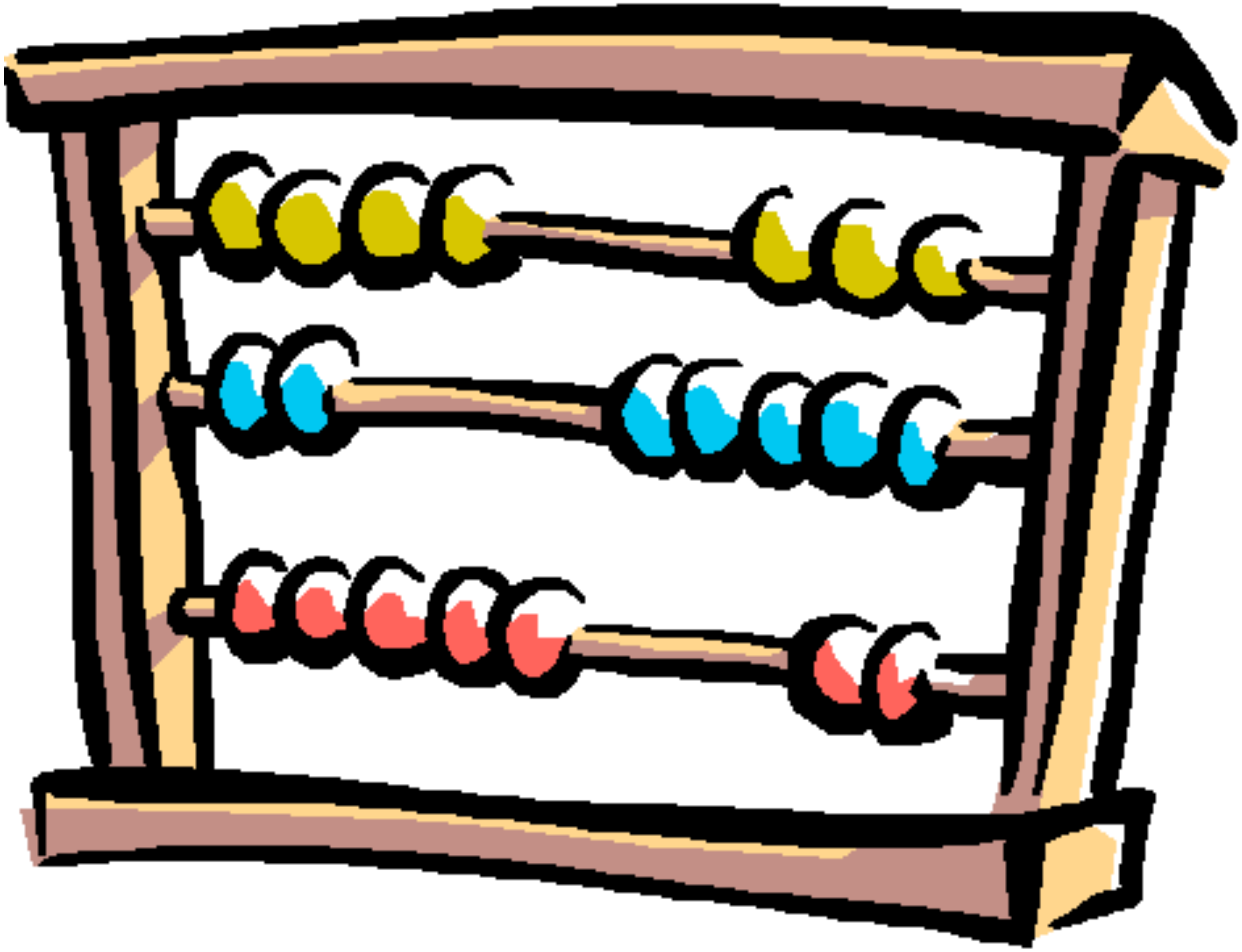
```
?- numeral(X).
```

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



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
```

Search tree

Draw the
search tree!

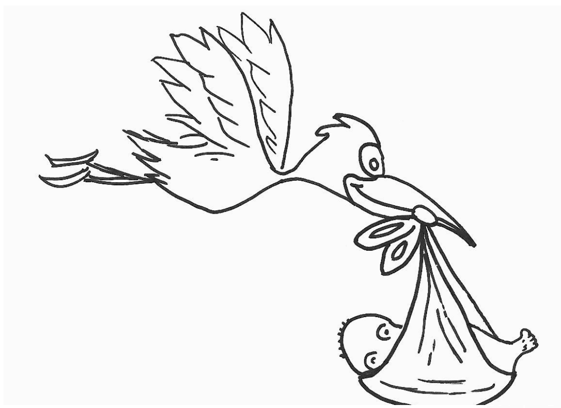
Prolog and Logic

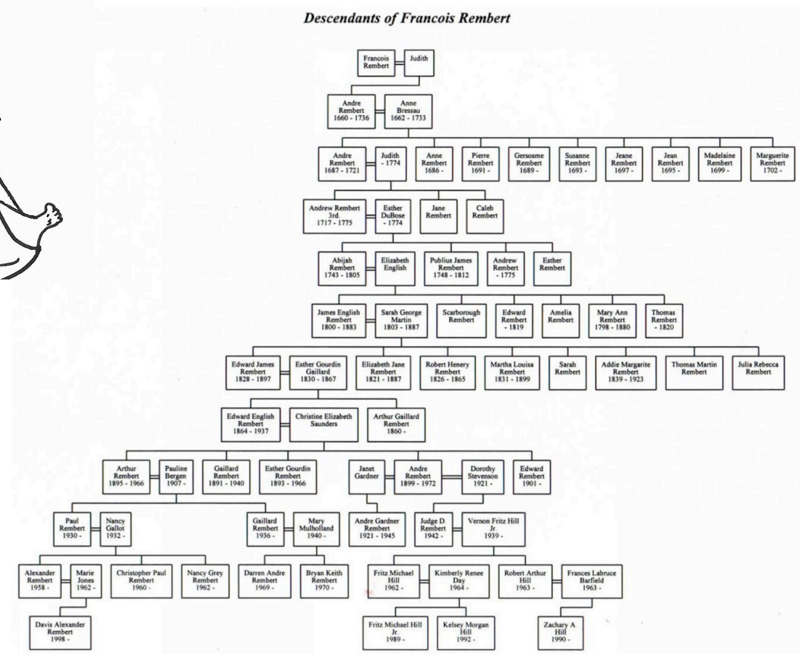
- Prolog was the first reasonable attempt to create a logic programming language
  - Programmer gives a declarative specification of the problem, using the language of logic
  - The programmer should not have to tell the computer what to do
  - To get information, the programmer simply asks a query

Prolog and Logic

- Prolog does some important steps in this direction
- 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

Four different descend/2





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).
```

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
```

FIRST SOLUTION

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).
```

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
```

FIRST SOLUTION

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).
```

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).
```

HOW MANY SOLUTIONS WILL THIS QUERY GENERATE BEFORE RUNNING OUT OF MEMORY?

Summary of this lecture

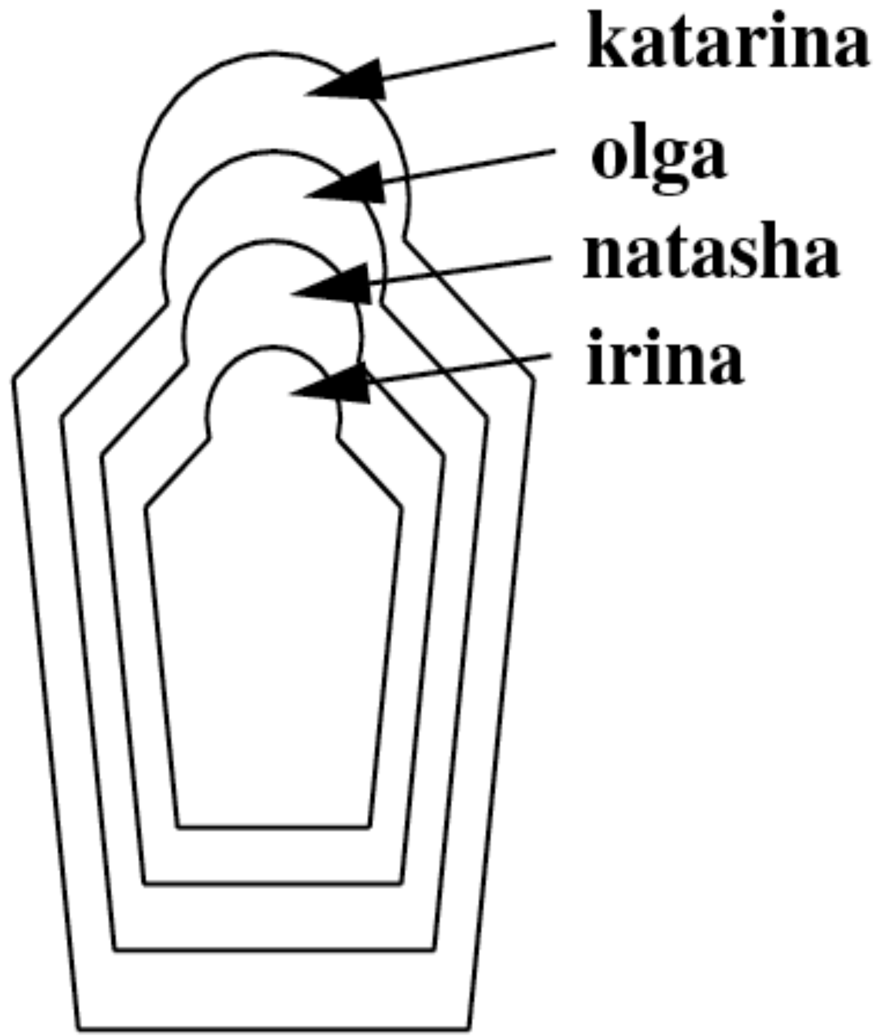
- In this lecture we introduced recursive predicates
- We also looked at the differences between the declarative and the procedural meaning of Prolog programs
- We have identified some of the shortcomings of Prolog seen as a logical programming language

Exercise 3.2: Matryoshka dolls

Exercise 3.2: Matryoshka dolls

First, write a knowledge base using the predicate directlyIn/2 which encodes which doll is directly contained in which other doll.

Then, define a recursive predicate in/2, that tells us which doll is (directly or indirectly) contained in which other dolls.



Next lecture

- Introduce **lists** in Prolog
  - Important recursive data structure in Prolog programming
  - Define the member/2 predicate, a fundamental Prolog tool for working with lists
  - Discuss the idea of recursing down lists