Lecture 3: Recursion

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

- Exercises
  - Corrections exercises LPN chapter 2
  - Exercises of LPN chapter 3

Recursive Definitions

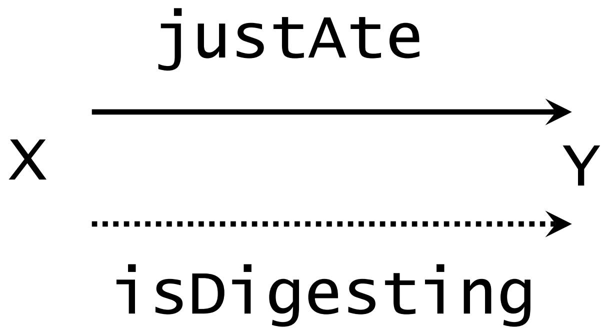
- Prolog predicates can be defined recursively
- A predicate is recursively defined if one or more rules in its definition refers to itself

Example 1: Eating

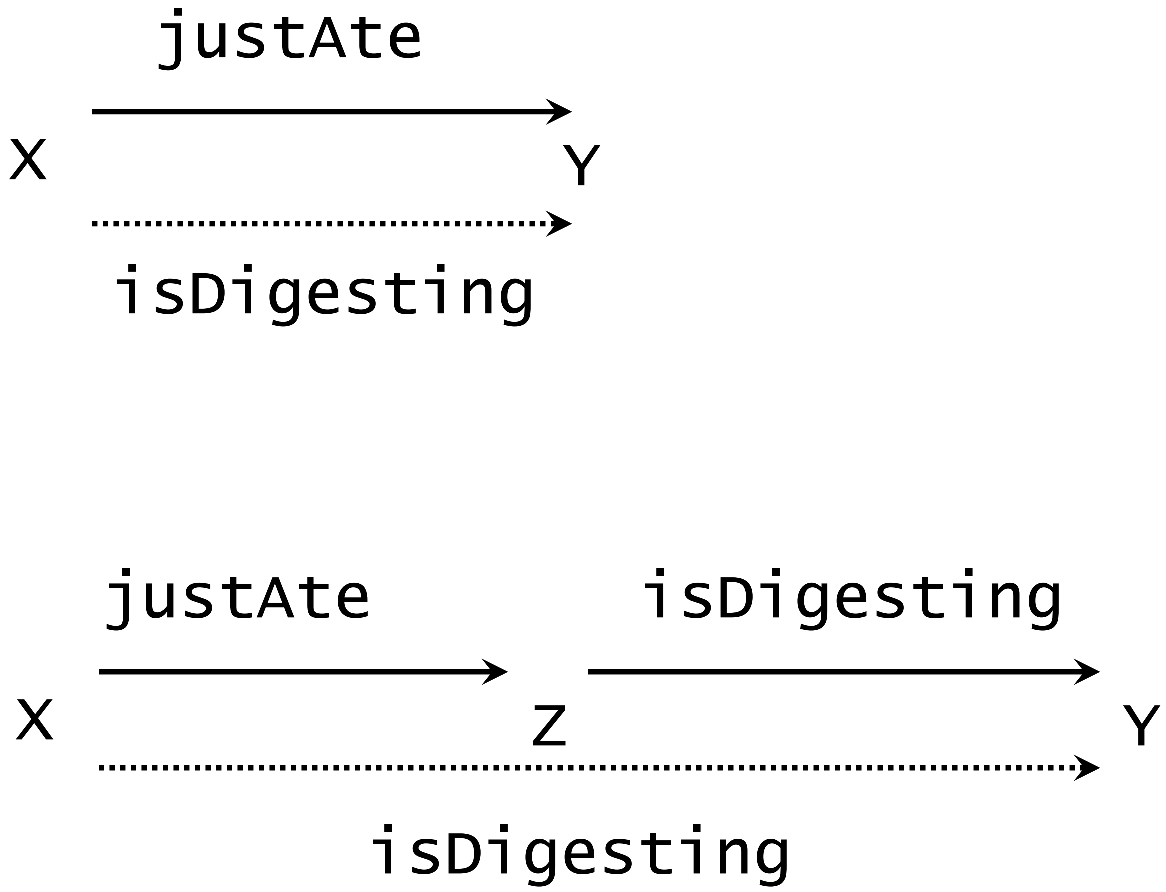
```
isDigesting(X,Y):- justAte(X,Y).
isDigesting(X,Y):- justAte(X,Z), isDigesting(Z,Y).

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).
```

```
?-
```

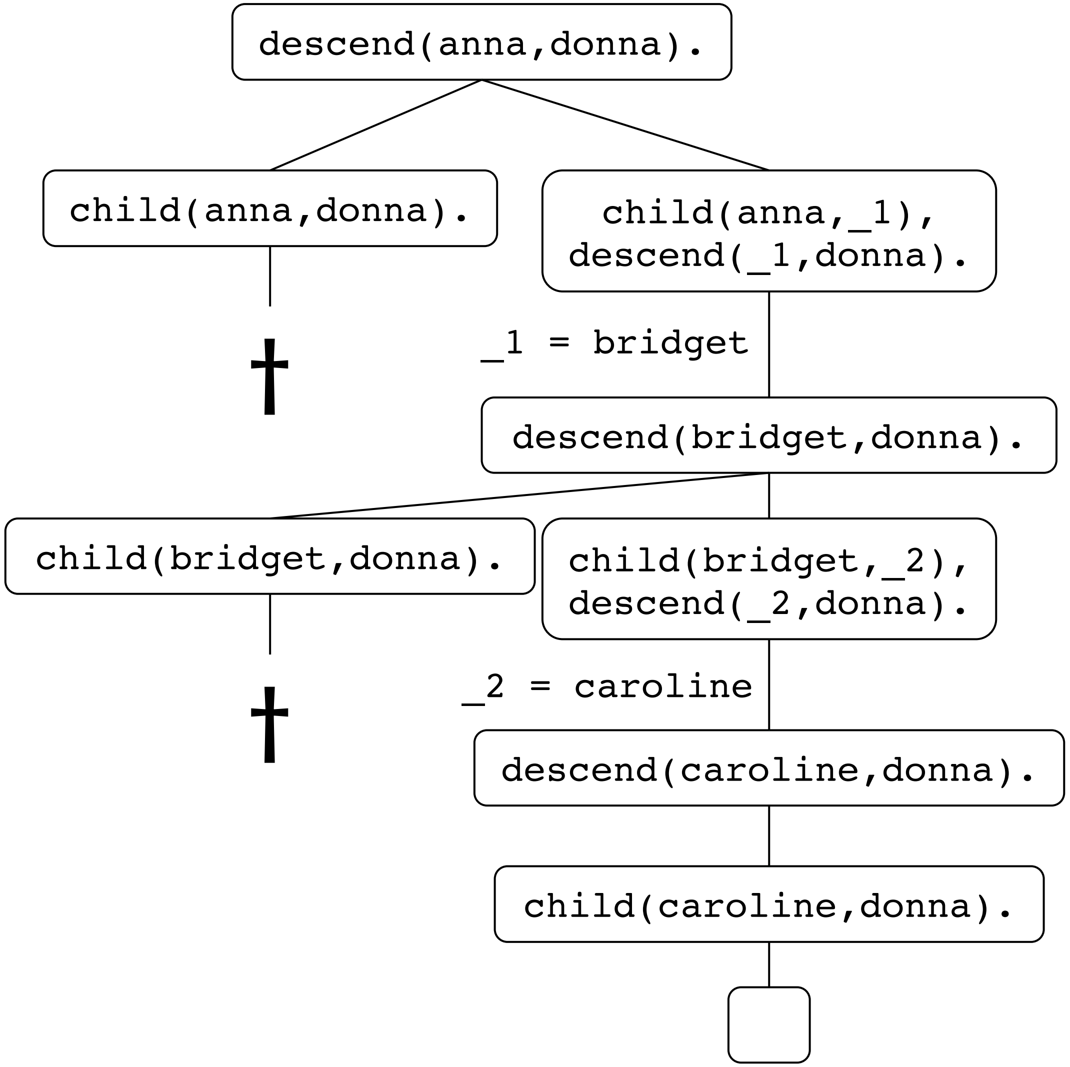
Example 2: Decendant

```
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 2: Search tree for `descend(anna,donna)`

```
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 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
```

Example 4: Search tree

```
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))), R).
        R = succ(_1)                          R = succ(_1)
add(succ(0), succ(succ(succ(0))), _1).
        _1 = succ(_2)                         R = succ(succ(_2))
add(0, succ(succ(succ(0))), _2).
        _2 = succ(succ(succ(0)))              R = succ(succ(succ(succ(succ(0)))))
add(0,succ(succ(succ(0))),succ(succ(succ(0)))).
        □
```

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

Correction Exercise 2.1
Recall Unification Definition

1. If T_1 and T_2 are constants, then T_1 and T_2 unify if they are the same atom, or the same number.
2. If T_1 is a variable and T_2 is any type of term, then T_1 and T_2 unify, and T_1 is instantiated to T_2. (and vice versa)
3. If T_1 and T_2 are complex terms then they unify if:
   a) They have the same functor and arity, and
   b) all their corresponding arguments unify, and
   c) the variable instantiations are compatible.

Correction Exercise 2.1

1. bread = bread ✔
2. 'Bread' = bread ✗
3. 'bread' = bread ✔
4. Bread = bread ✔
5. bread = sausage ✗
6. food(bread) = bread ✗
7. food(bread) = X ✔
8. food(X) = food(bread) ✔
9. food(bread,X) = food(Y,sausage) ✔
10. food(bread,X,beer) = food(Y,sausage,X) ✗
11. food(bread,X,beer) = food(Y,kahuna_burger) ✗
12. food(X) = X ✔
13. meal(food(bread),drink(beer)) = meal(X,Y) ✔
14. meal(food(bread),X) = meal(X,drink(beer)) ✗

Correction Exercise 2.2

```
house_elf(dobby).
witch(hermione).
witch('McGonagall').
witch(rita_skeeter).
magic(X) :- house_elf(X).
magic(X) :- wizard(X).
magic(X) :- witch(X).
```

```
?- magic(house_elf).
false.
%%% actually raises exception: wizard/1 not defined
```

Correction Exercise 2.2

```
house_elf(dobby).
witch(hermione).
witch('McGonagall').
witch(rita_skeeter).
magic(X) :- house_elf(X).
magic(X) :- wizard(X).
magic(X) :- witch(X).
```

```
?- wizard(harry).
false.
%%% actually raises exception: wizard/1 not defined
```

Correction Exercise 2.2

```
house_elf(dobby).
witch(hermione).
witch('McGonagall').
witch(rita_skeeter).
magic(X) :- house_elf(X).
magic(X) :- wizard(X).
magic(X) :- witch(X).
```

```
?- magic(wizard).
false.
%%% actually raises exception: wizard/1 not defined
```

Correction Exercise 2.2

```
house_elf(dobby).
witch(hermione).
witch('McGonagall').
witch(rita_skeeter).
magic(X) :- house_elf(X).
magic(X) :- wizard(X).
magic(X) :- witch(X).
```

```
?- magic('McGonagall').
true.
%%% actually raises exception: wizard/1 not defined
```

Correction Exercise 2.2

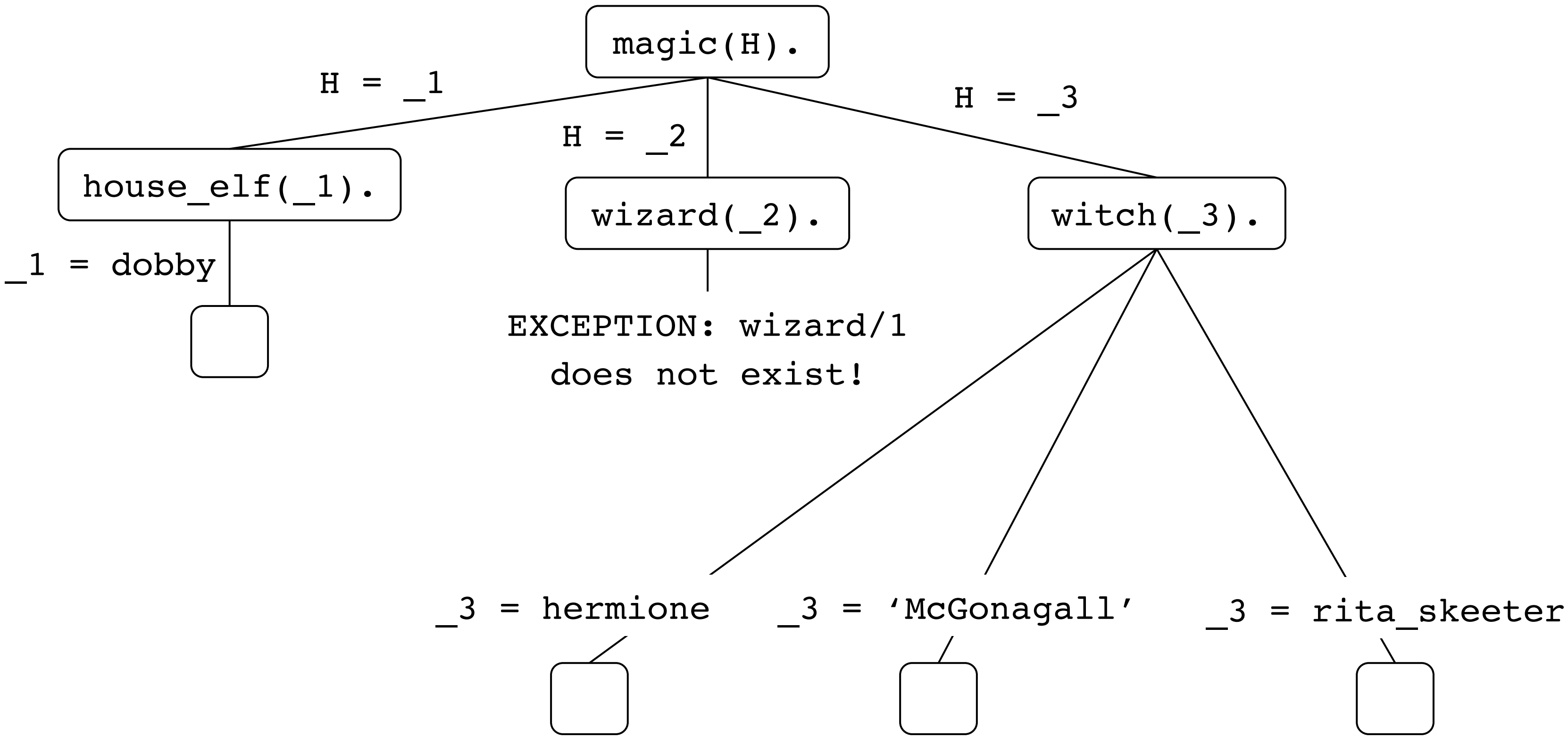
```
house_elf(dobby).
witch(hermione).
witch('McGonagall').
witch(rita_skeeter).
magic(X) :- house_elf(X).
magic(X) :- wizard(X).
magic(X) :- witch(X).
```

```
?- magic(Hermione).
Hermione = dobby;
%%% actually raises exception: wizard/1 not defined
Hermione = hermione;
Hermione = ‘McGonagall’;
Hermione = rita_skeeter;
```

Correction Exercise 2.2

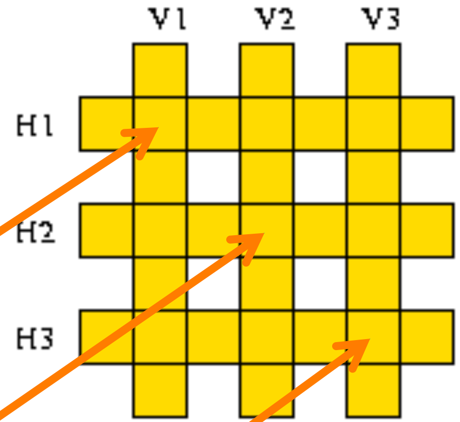
```
house_elf(dobby).
witch(hermione).
witch('McGonagall').
witch(rita_skeeter).
```

```
magic(X) :- house_elf(X).
magic(X) :- wizard(X).
magic(X) :- witch(X).
```



Correction Exercise 2.4

```
word(abalone,a,b,a,l,o,n,e).
word(abandon,a,b,a,n,d,o,n).
word(enhance,e,n,h,a,n,c,e).
word(anagram,a,n,a,g,r,a,m).
word(connect,c,o,n,n,e,c,t).
word(elegant,e,l,e,g,a,n,t).



crosswd(V1,V2,V3,H1,H2,H3) :-
   word(V1,_,V1H1,_,V1H2,_,V1H3,_),
   word(V2,_,V2H1,_,V2H2,_,V2H3,_),
   word(V3,_,V3H1,_,V3H2,_,V3H3,_),
   word(H1,_,V1H1,_,V1H2,_,V1H3,_),
   word(H2,_,V2H1,_,V2H2,_,V2H3,_),
   word(H3,_,V3H1,_,V3H2,_,V3H3,_).
```



Beware: this allows to use a word more than once!!

Exercises Chapter 3

- 3.2, 3.3, 3.4

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