CPSC 312

Functional and Logic Programming

22 September 2015

Assignment 1

- Due next Tuesday at the beginning of the class. I'll be collecting hard copies at the start of the class.
- You can also hand it in today (after the class), on Thursday after or before the class or slide it under my office door (ICCS 187) anytime before noon next Tuesday.
- Today is the last day to drop without a W mark.

Midterm

Thursday

October 15th

during class time

Office Hours

- Instructor: (Sara Sagaii) sarams@cs.ubc.ca
  - Tuesdays 10-11am
  - Thursdays 10:30-11:30am
  - ICCS 187
- Rui Ge:
  - Wednesdays 10-12am
  - Table 1 at DLC
- Susanne Bradley:
  - Fridays 11:30-1:30
  - Table 1 at DLC

Questions

Review

- Last time we talked about a few special cases with prolog.
- We looked at the resolution algorithm again, this time with rules in the program.
- We also talked about nondeterminism, its impracticality, and how certain theoretical promises of logic programming are compromised in practice.

Simple Abstract Interpreter

Input: A ground **goal** `G` and a **program** P

Output: *yes* if `G` is a **logical consequence** of P,
no otherwise

Algorithm:

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
   choose a goal A from the resolvent
   choose a ground instance of a clause
         A' :- $B_1$,…,$B_n$ from program P
         such that A and A' are identical
         (if no such goal and clause exist, exit
             the while loop)
    replace A by $B_1$,…,$B_n$ in the resolvent
If the resolvent is empty, then output yes,
   else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).
resolvent=
grandmother(betty,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose a ground instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' are identical
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output  yes,
else output no
```

OOPS! Time to modify things a bit.

Unification

Matching a goal in the resolvent with the head of a rule in program P is the heart of logic programming or automated theorem proving.

The matching is what allows the body of the chosen rule to replace the chosen goal in the resolvent – that's the inference step. No matching => no inference!

Matching is very easy when there are no variables involved, as just noted. But when the terms we're trying to match contain variables, the matching process becomes complicated.

Unification

The general purpose matching process that is used to match terms is called unification, which works for both ground and non-ground cases.

The goal of unification is to find a substitution that makes two terms (a goal in the resolvent and the head of some rule in program P) identical.

Such a substitution is called a unifier and is denoted by the symbol θ. A unifier is just a list of equivalences that mean “you can substitute this for that”. The *mgu* is the most general unifier, meaning the associated common instance resulting from it is the most general.

Unification Algorithm

Input: Two terms T1 and T2 to be unified
Output: θ, the mgu of T1 and T2, or failure

Algorithm: Initialize the substitution θ to be empty, the stack to contain the equation T1 = T2,

and failure to false. And then ----->

```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to $\theta$

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to $\theta$

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output $\theta$.
```

Example

Initial state:

T1 = grandmother(betty,roxy)

T2 = grandmother(X,Y)

failure = false

$\theta = \{\}$

Initialize the stack to contain T1 = T2 :

grandmother(betty,roxy) =

grandmother(X,Y)

```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to θ

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to θ

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output θ.
```

Example

Initial state:

T1 = grandmother(betty,roxy)

T2 = grandmother(X,Y)

failure = false

θ = {}

Initialize the stack to contain T1 = T2 :

grandmother(betty,roxy) = grandmother(X,Y)

```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to θ

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to θ

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output θ.
```

Example

Initial state:

T1 = grandmother(betty,roxy)

T2 = grandmother(X,Y)

failure = false

θ = {}

Initialize the stack to contain T1 = T2 :

grandmother(betty,roxy) =

grandmother(X,Y)

```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to θ

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to θ

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output θ.
```

Example

T1 = grandmother(betty,roxy)

T2 = grandmother(X,Y)

failure = false

θ = {}

grandmother(betty,roxy) =

grandmother(X,Y)

```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to $\theta$

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to $\theta$

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output $\theta$.
```

Example

T1 = grandmother(betty,roxy)

T2 = grandmother(X,Y)

failure = false

θ = {}

grandmother(betty,roxy) =

grandmother(X,Y)

```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to $\theta$

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to $\theta$

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output $\theta$.
```

Example

T1 = grandmother(betty,roxy)

T2 = grandmother(X,Y)

failure = false

θ = {}

grandmother(betty,roxy) =

grandmother(X,Y)

betty = X

roxy = Y

```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to $\theta$

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to $\theta$

   X and Y are identical constants or variables:
      continue
   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output $\theta$.
```

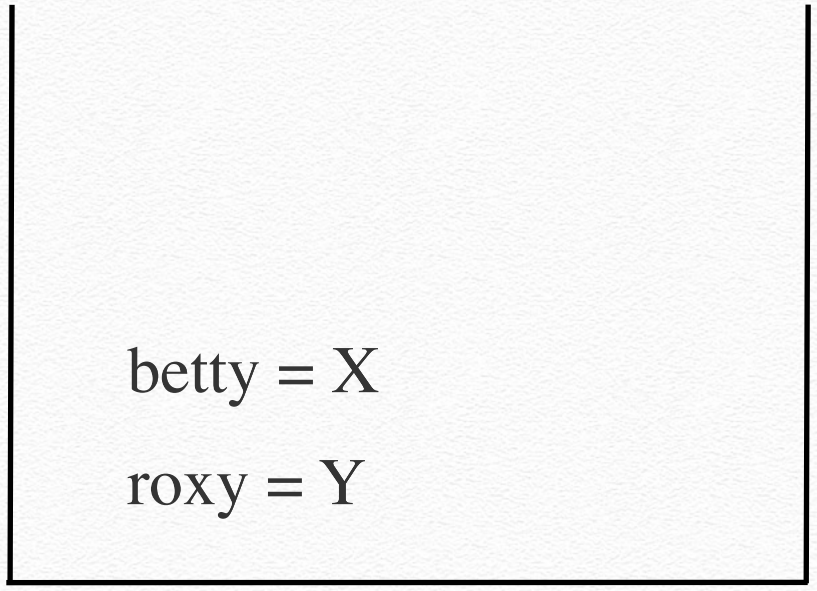
Example

T1 = grandmother(betty,roxy)

T2 = grandmother(X,Y)

failure = false

θ = {}



```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
       substitute Y for X in the stack
       and add X = Y to $\theta$

   Y is a variable that does not occur in X:
       substitute X for Y in the stack
       and add Y = X to $\theta$

   X and Y are identical constants or variables:
       continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
       functor f and n > 0:
       push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output $\theta$.
```

Example

T1 = grandmother(betty,roxy)

T2 = grandmother(X,Y)

failure = false

θ = {}

betty = X

roxy = Y

```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to θ

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to θ

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output θ.
```

Example

T1 = grandmother(betty,roxy)

T2 = grandmother(X,Y)

failure = false

θ = {}

betty = X

roxy = Y

```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to $\theta$

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to $\theta$

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output $\theta$.
```

Example

T1 = grandmother(betty,roxy)

T2 = grandmother(X,Y)

failure = false

$\theta = \{\}$

betty = X

roxy = Y

```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to $\theta$

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to $\theta$

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output $\theta$.
```

Example

T1 = grandmother(betty,roxy)

T2 = grandmother(X,Y)

failure = false

θ = {}

betty = X

roxy = Y

```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to θ

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to θ

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output θ.
```

Example

T1 = grandmother(betty,roxy)

T2 = grandmother(X,Y)

failure = false

θ = {X=betty}

betty = X

roxy = Y

```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to θ

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to θ

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output θ.
```

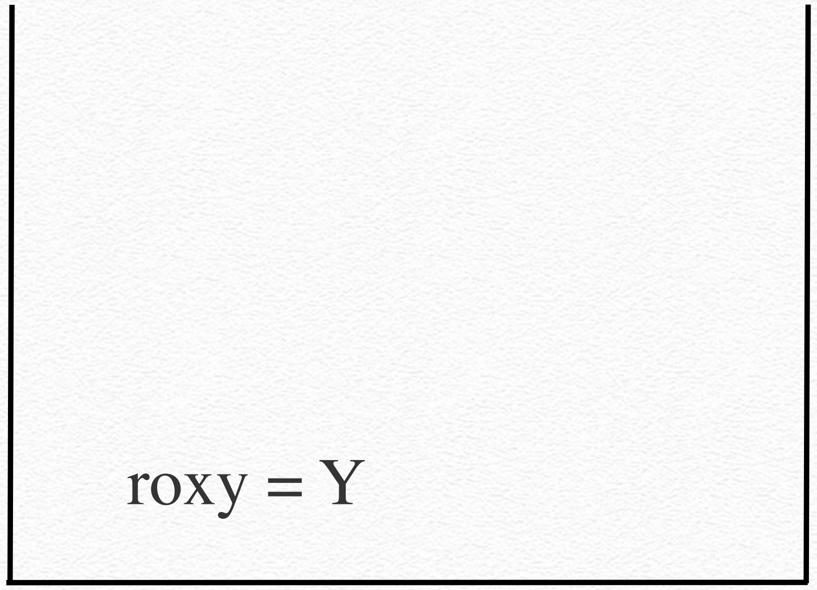
Example

T1 = grandmother(betty,roxy)

T2 = grandmother(X,Y)

failure = false

θ = {X=betty}



```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to θ

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to θ

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output θ.
```

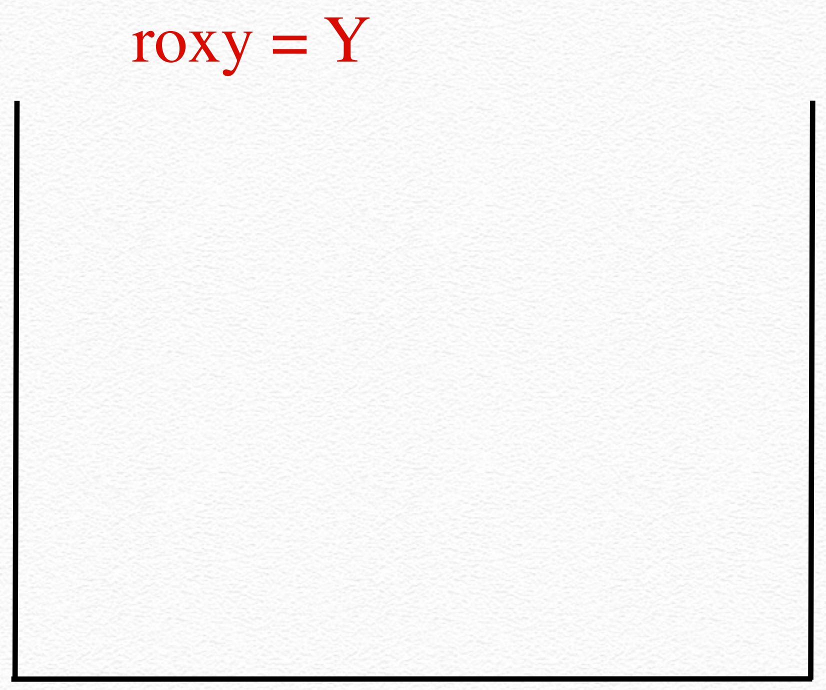
Example

T1 = grandmother(betty,roxy)

T2 = grandmother(X,Y)

failure = false

θ = {X=betty}



```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to θ

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to θ

   X and Y are identical constants or variables:
      continue
   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output θ.
```

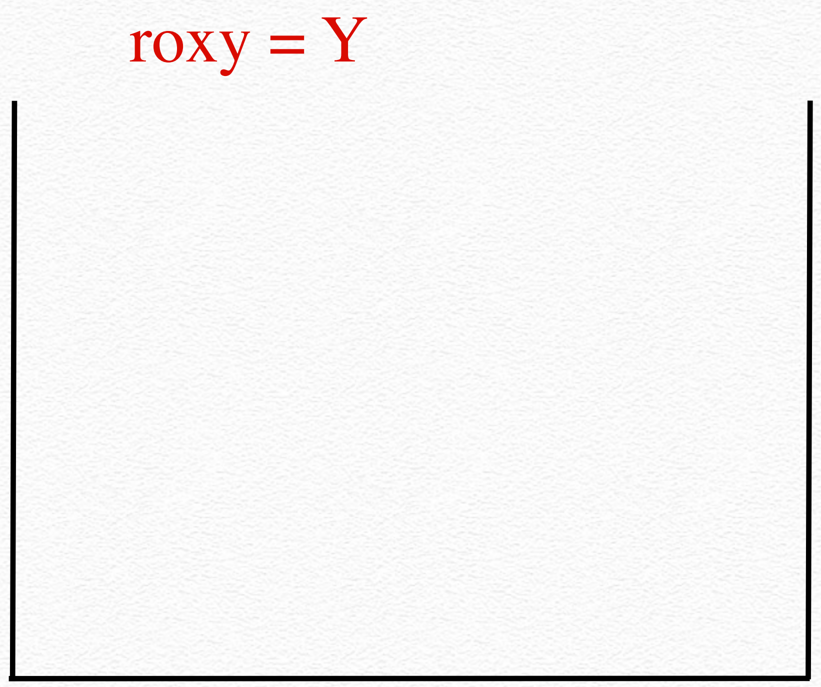
Example

T1 = grandmother(betty,roxy)

T2 = grandmother(X,Y)

failure = false

θ = {X=betty}



```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to $\theta$

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to $\theta$

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output $\theta$.
```

Example

T1 = grandmother(betty,roxy)

T2 = grandmother(X,Y)

failure = false

θ = {X=betty}

roxy = Y

```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to θ

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to θ

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output θ.
```

Example

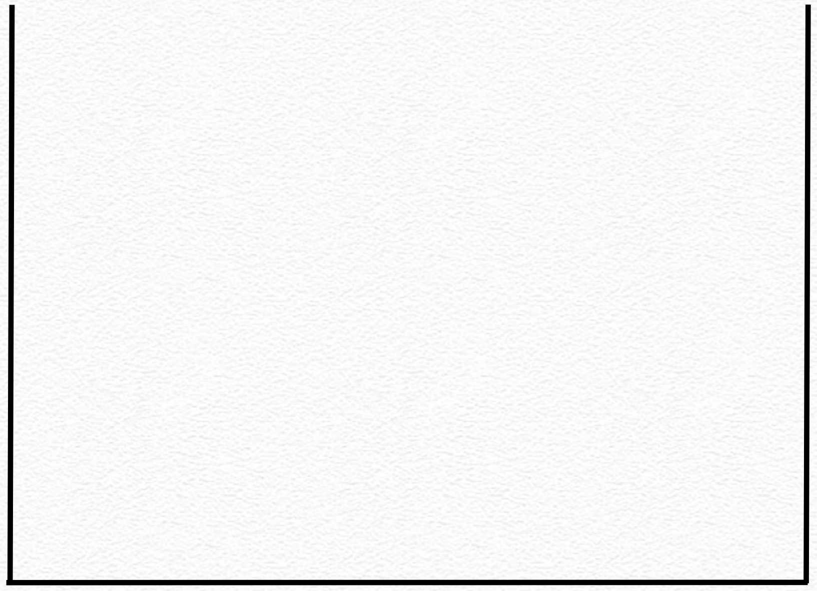
T1 = grandmother(betty,roxy)

T2 = grandmother(X,Y)

failure = false

θ = {X=betty,Y = roxy}

roxy = Y



```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to $\theta$

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to $\theta$

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output $\theta$.
```

Example

T1 = grandmother(betty,roxy)

T2 = grandmother(X,Y)

failure = false

θ = {X=betty,Y = roxy}

```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to $\theta$

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to $\theta$

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output $\theta$.
```

Example

T1 = grandmother(betty,roxy)

T2 = grandmother(X,Y)

failure = false

θ = {X=betty,Y = roxy}

```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to $\theta$

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to $\theta$

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output $\theta$.
```

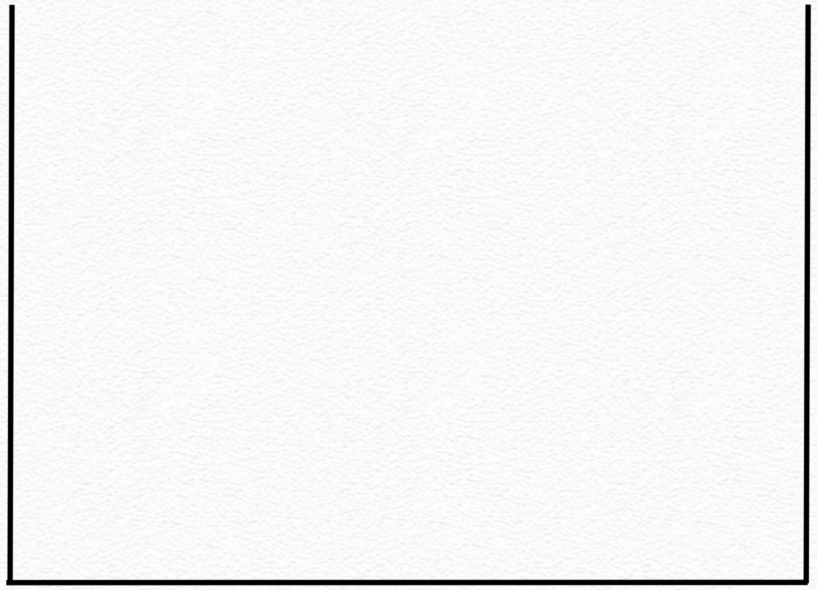
Example

T1 = grandmother(betty,roxy)

T2 = grandmother(betty,roxy)

failure = false

θ = {X=betty,Y = roxy} <-- The Output



```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to $\theta$

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to $\theta$

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output $\theta$.
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).

resolvent=
grandmother(betty,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose a ground instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' are identical
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output  yes,
else output no
```

OOPS! Time to modify things a bit.

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).
resolvent=
grandmother(betty,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output   yes,
else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).
```

$\theta = \{X=betty, Y = roxy\}$

```
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).
resolvent=
grandmother(betty,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output  yes,
else output no
```

What if P has rules?

The program (P):

```
father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(betty,roxy) :-
mother(betty,Z),parent(Z,rox
y).
```

θ = {X=betty,Y = roxy}

```
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).
resolvent=
grandmother(betty,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output  yes,
else output no
```

Example

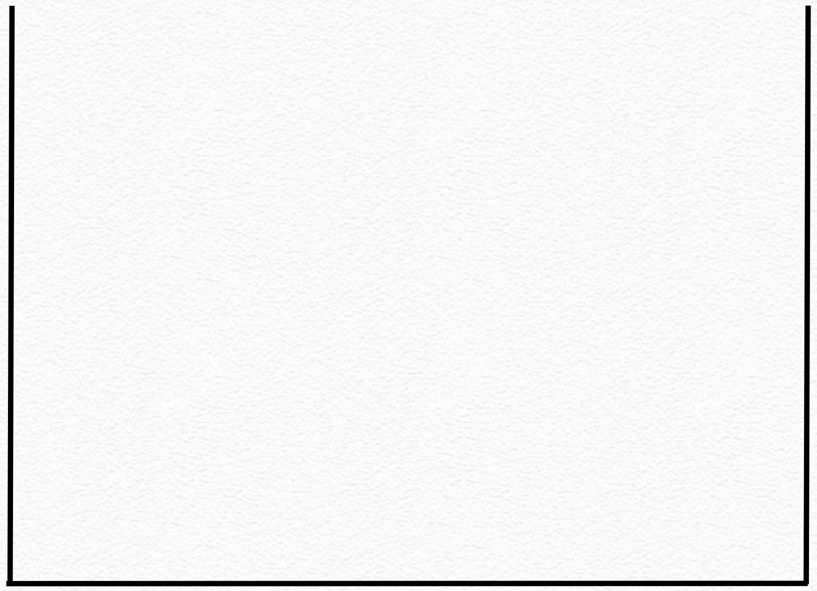
What if the output was failure?

T1 = grandmother(betty,roxy)

T2 = grandmother(betty,roxy)

failure = true <-- The Output?

θ = {X=betty,Y = roxy}



Example

T1 = grandmother(betty,roxy)

T2 = grandmother(betty,roxy)

failure = true <-- The Output?

θ = {X=betty,Y = roxy}

What if the output was failure?

To the "abstract interpreter" failure to unify means failure in resolution.

Example

T1 = grandmother(betty,roxy)

T2 = grandmother(betty,roxy)

failure = true <-- The Output?

θ = {X=betty,Y = roxy}

What if the output was failure?

To the "abstract interpreter" failure to unify means failure in resolution.

But to a sequential Prolog interpreter such as SWI, failure simply means, move on to the next clause, unless there are no more clauses left.

Prolog vs the abstract interpreter

"The choice of goal to reduce is arbitrary; it does not matter which is chosen for the computation to succeed. If there is a successful computation by choosing a given goal, then there is a successful computation by choosing any other goal."

"The choice of the clause to effect the reduction is nondeterministic. Not every choice will lead to a successful computation. A nondeterministic interpreter "guesses" the clause that leads to a successful computation and chooses it."

Prolog vs the abstract interpreter

How does one implement nondeterminism?

find a psychic

explore all possible solutions in parallel

depth-first search with backtracking (SWI-Prolog)

True nondeterminism cannot be realized but can be simulated or approximated.

Prolog's execution model

"Prolog's execution mechanism is obtained from the abstract interpreter by choosing the leftmost goal instead of an arbitrary one and replacing the nondeterministic choice of a clause by sequential search for a unifiable clause and backtracking"

"Prolog simulates the nondeterministic choice of reducing clause by sequential search and backtracking. When attempting to reduce a goal, the first clause whose head unifies with the goal is chosen. If no unifiable clause is found for the popped goal, the computation is unwound to the last choice made, and the next unifiable clause is chosen."

What if P has rules?

The program (P):

```
father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).
```

```
grandmother(betty,roxy) :-
mother(betty,Z),parent(Z,rox
y).
```

θ = {X=betty,Y = roxy}

```
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).
resolvent=
grandmother(betty,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output   yes,
else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).


parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).
resolvent= parent(bamm-
bamm,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output yes,
else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).

which one to choose?
```

```
?- grandmother(betty,roxy).

resolvent= parent(bamm-
bamm,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output  yes,
else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

the abstract interpreter will choose the right one

```
?- grandmother(betty,roxy).

resolvent= parent(bamm-bamm,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output  yes,
else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

SWI-Prolog: sequential search. choose the first one.

```
?- grandmother(betty,roxy).

resolvent= parent(bamm-bamm,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output  yes,
else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

SWI-Prolog: sequential search. choose the first one. REMEMBER THE CHOICE

```
?- grandmother(betty,roxy).

resolvent= parent(bamm-bamm,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output yes,
else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

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

$\theta = \{X=\text{bamm-bamm}, Y = \text{roxy}\}$

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

```
?- grandmother(betty,roxy).

resolvent= parent(bamm-
bamm,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
    choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output yes,
else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(bamm-bamm,roxy) :-
mother(bamm-bamm,roxy).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).

resolvent= parent(bamm-
bamm,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output yes,
else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(bamm-bamm,roxy) :-
mother(bamm-bamm,roxy).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).

resolvent= parent(bamm-
bamm,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output yes,
else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(bamm-bamm,roxy) :-
mother(bamm-bamm,roxy).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).

resolvent= mother(bamm-
bamm,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output  yes,
else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).

resolvent= mother(bamm-
bamm,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output yes,
else output no
```

What if P has rules?

The program (P):

```
father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).

resolvent= mother(bamm-
bamm,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output yes,
else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).

resolvent= mother(bamm-
bamm,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output yes,
else output no
```

What if P has rules?

The program (P):

```
father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).

resolvent= mother(bamm-
bamm,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output yes,
else output no
```

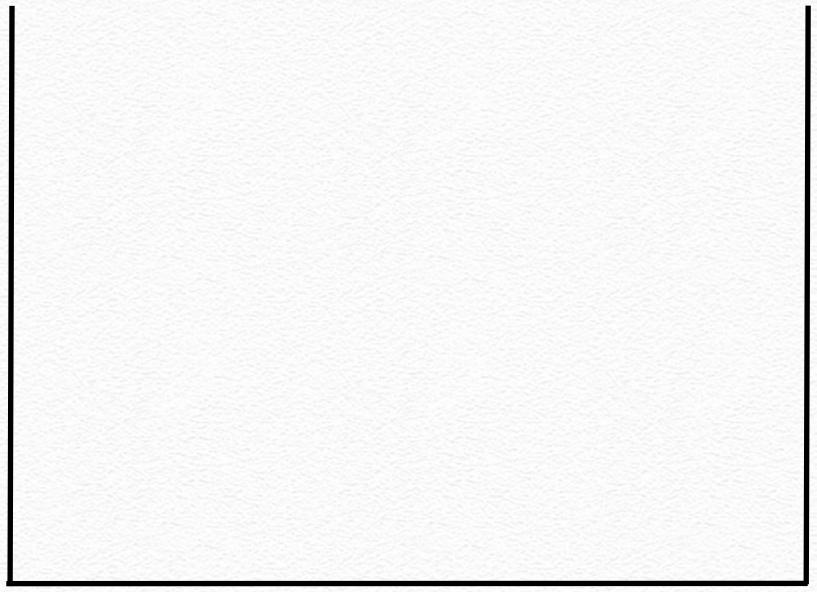
Example

T1 = mother(bamm-bamm,roxy)

T2 = mother(pebbles,roxy)

failure = false

$\theta = \{\}$



```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to $\theta$

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to $\theta$

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output $\theta$.
```

Example

T1 = mother(bamm-bamm,roxy)

T2 = mother(pebbles,roxy)

failure = false

θ = {}

bamm-bamm=pebbles

roxy=roxy

```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to $\theta$

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to $\theta$

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output $\theta$.
```

Example

T1 = mother(bamm-bamm,roxy)

T2 = mother(pebbles,roxy)

failure = false

θ = {}

bamm-bamm=pebbles

roxy=roxy

```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to θ

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to θ

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output θ.
```

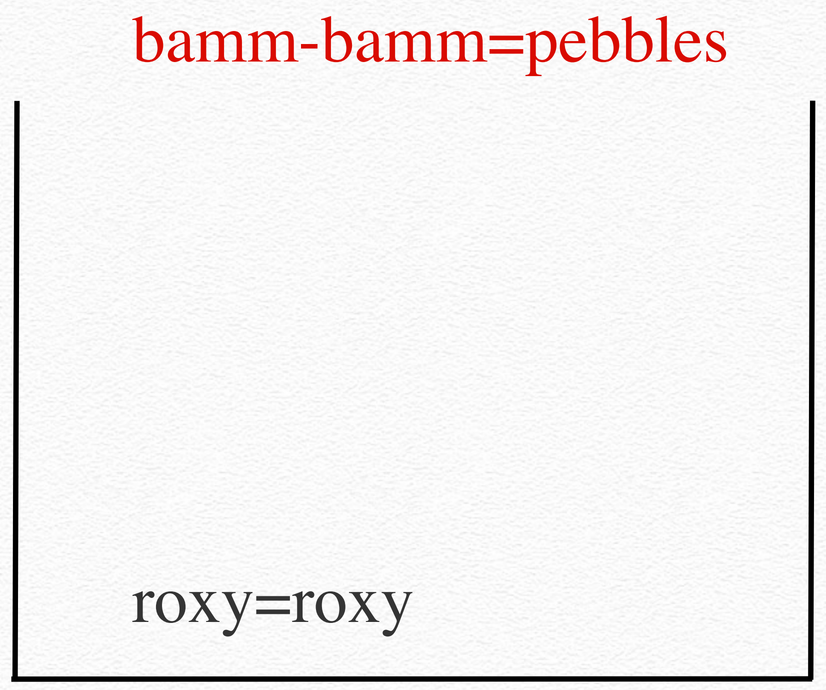
Example

T1 = mother(bamm-bamm,roxy)

T2 = mother(pebbles,roxy)

failure = false

θ = {}



```
while stack not empty and no failure do

 pop X = Y from the stack

 case
   X is a variable that does not occur in Y:
      substitute Y for X in the stack
      and add X = Y to θ

   Y is a variable that does not occur in X:
      substitute X for Y in the stack
      and add Y = X to θ

   X and Y are identical constants or variables:
      continue

   X is f(X1,...,Xn) and Y is f(Y1,...,Yn) for some
      functor f and n > 0:
      push Xi = Yi, i = 1...n, on the stack

   otherwise: failure is true

If failure, then output failure else output θ.
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).

resolvent= mother(bamm-
bamm,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output  yes,
else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).

resolvent= mother(bamm-
bamm,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
    choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output  yes,
else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

Remember the choice? Backtrack to that point.

```
?- grandmother(betty,roxy).

resolvent= mother(bamm-
bamm,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
    choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output  yes,
else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

SWI-Prolog: sequential search. choose the first one. REMEMBER THE CHOICE

```
?- grandmother(betty,roxy).

resolvent= parent(bamm-
bamm,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output  yes,
else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).

resolvent= parent(bamm-
bamm,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
If the resolvent is empty, then output yes,
else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).
resolvent= father(bamm-
bamm,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
    choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
    (after performing the appropriate
    substitution)
If the resolvent is empty, then output yes,
else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).
resolvent= father(bamm-
bamm,roxy)
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
    (after performing the appropriate
substitution)
If the resolvent is empty, then output  yes,
else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).
resolvent=
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
    choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
    (after performing the appropriate
substitution)
If the resolvent is empty, then output yes,
else output no
```

What if P has rules?

```
The program (P):

father(fred,pebbles).
father(bamm-bamm,roxy).
father(barney,bamm-bamm).
father(bamm-bamm,chip).
mother(pebbles,roxy).
mother(pebbles,chip).
mother(wilma,pebbles).
mother(betty,bamm-bamm).

grandmother(X,Y) :-
mother(X,Z),parent(Z,Y).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

```
?- grandmother(betty,roxy).
yes.
```

```
Initialize resolvent to goal G (the query)
while resolvent not empty do
    choose a goal A from the resolvent
    choose an instance of a clause
        A' :- B1,...,Bn from program P
        such that A and A' unify
        (if no such goal and clause exist,
    exit the while loop)
    replace A by B1,...,Bn in the resolvent
    (after performing the appropriate
substitution)
If the resolvent is empty, then output  yes,
else output no
```

Questions

Where declarative fails

"Ideally one should write axioms that define the desired relations, maintaining ignorance of the way they are going to be used by the execution mechanism. "

Consequences for the programmer

- Chapter 7 specifies the consequences of Prolog's execution model for the logic programmer. Another way to understand them is as best practices in prolog programming.
- We've talked about most of them and you're doing an assignment on it. It is a very important chapter and reading it will help you with your assignment as well. So read it. These chapters on Prolog on written in a much simpler language than the first 5 chapters that are on logic programming.
- Rule Order
- Goal Order
- Termination
- Redundant Solutions

Questions?

List: A recursive data structure

- An ordered sequence of elements:

```
[harpo, groucho, chico]
```

- Consists of two parts: head and tail

```
[harpo | [groucho, chico]]
```

List: A recursive data structure

the “root” of all lists is the empty list: `[]`

for example

```
[harpo | [groucho | [chico | [] ]]]
```

List: A recursive data structure

Lists are defined recursively: A list is either the empty list or a two element structure whose first element is called the head and whose second element is itself a list, called either the tail or the rest:

```
list([]).
list(X|Xs) :- list(Xs).
```

Variation in syntax

dot functor syntax: `.(a, .(b, .(c, [])))`

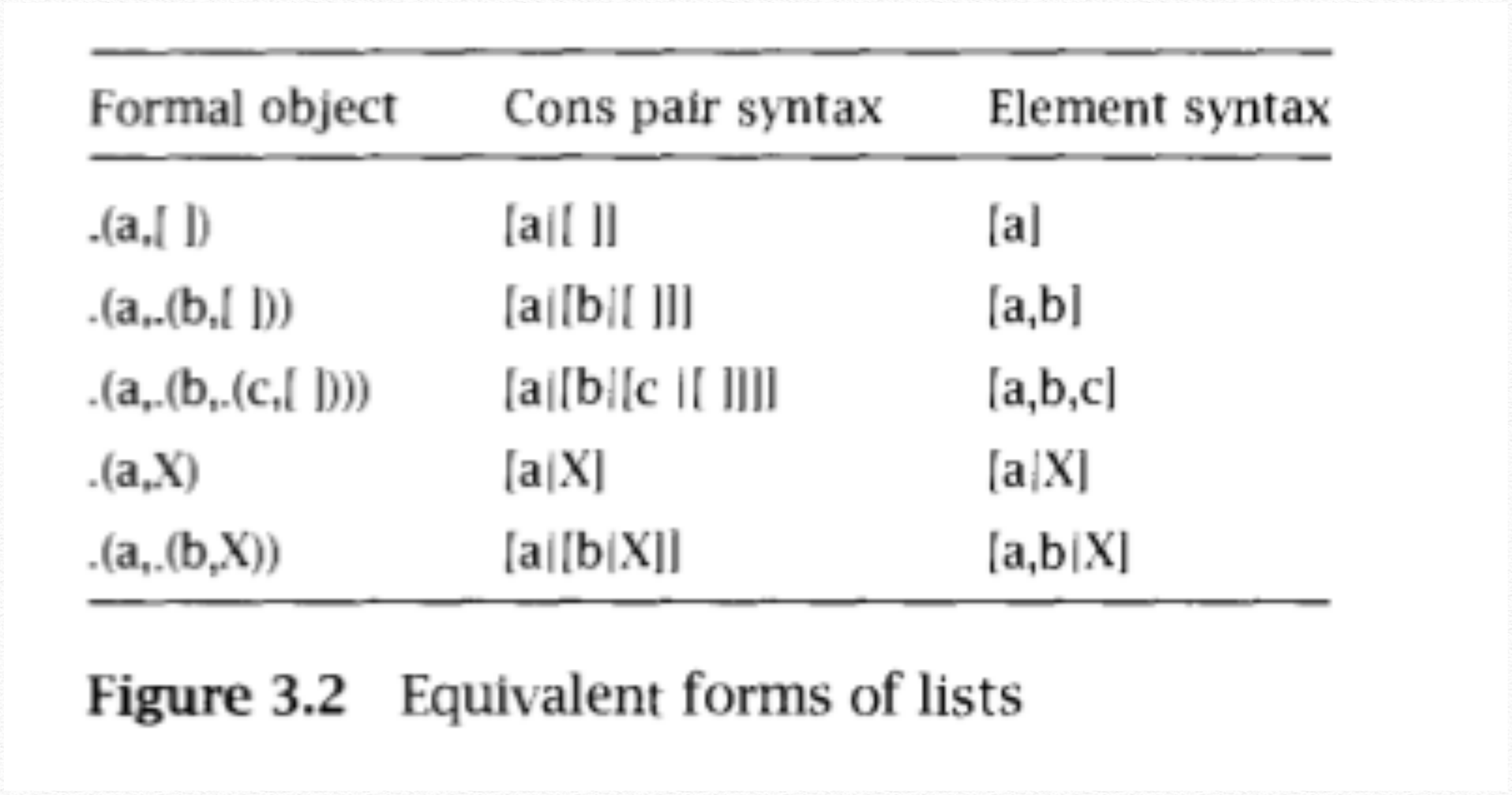
Cons pair syntax : `[a | [b | [c | [ ]]]]`

Element syntax : `[a,b,c]`

graphical notation: binary tree

Note that element syntax is purely cosmetic. Lists are stored as binary trees, reflected in the cons and the dot notation syntax, but when Prolog prints a list, it will use element syntax as much as possible to make it pretty.

Variation in syntax



Formal object	Cons pair syntax	Element syntax
.(a,[ ])	[a|[ ]]	[a]
.(a,.(b,[ ]))	[a|[b|[ ]]]	[a,b]
.(a,.(b,.(c,[ ])))	[a|[b|[c |[ ]]]]	[a,b,c]
.(a,X)	[a|X]	[a|X]
.(a,.(b,X))	[a|[b|X]]	[a,b|X]

Figure 3.2 Equivalent forms of lists

Recursive operation on List

The common beginning problem in list processing is how to determine if some item is a member of some list.

How do you do this in Prolog?

Recursively: X is an element of a list if it is either the first element of the list (the head) or if it is a member of the tail or the rest of the list.

List membership

X is an element of a list if it is either **the first element of the list (the head)** or if it is a member of the tail or the rest of the list.

```
member(X,[X|T]).
```

List membership

X is an element of a list if it is either the first element of the list (the head) or if it is a **member of the tail or the rest of the list**.

```
member(X,[X|T]).
member(X,[H|T]) :- member(X,T).
```

List membership: recursive solution

How could we arrive at this solution?

List membership: recursive solution

How could we arrive at this solution? We might have applied induction to some examples:

How do you prove `member(c,[a,b,c,d])`?

List membership: recursive solution

How could we arrive at this solution? We might have applied induction to some examples:

How do you prove `member(c,[a,b,c,d])`?

Prove `member(c,[b,c,d]).`

List membership: recursive solution

How could we arrive at this solution? We might have applied induction to some examples:

How do you prove `member(c,[a,b,c,d])`?

Prove `member(c,[b,c,d]).`

Prove `member(c,[c,d]).` That's easy.

List membership: recursive solution

How could we arrive at this solution? We might have applied induction to some examples:

How do you prove `member(c,[a,b,c,d])`?

Prove `member(c,[b,c,d])`.

Prove `member(c,[c,d])`. That's easy.

Therefore: `member(X,[X|T])`.

List membership: recursive solution

How could we arrive at this solution? We might have applied induction to some examples:

How do you prove `member(c,[a,b,c,d])`?

Prove `member(c,[b,c,d]).`

Prove `member(c,[c,d]).` That's easy.

Therefore: `member(X,[X|T]).`

What's the induction (and reduction) step?

List membership: recursive solution

How could we arrive at this solution? We might have applied induction to some examples:

How do you prove `member(c,[a,b,c,d])`?

Prove `member(c,[b,c,d]).`

Prove `member(c,[c,d]).` That's easy.

Therefore: `member(X,[X|T]).`

What's the induction (and reduction) step?

```
member(X,[H|T]) :- member(X,T).
```

List Processing

- Another list processing problem. How can Prolog prove that something is a list?
- How do you prove `list([a,b,c])`? The answer is Induction:

  Prove `list([b,c]).`

  Prove `list([c]).`

  Prove `list([]).` Well, that's easy, we just state that:

  ```
  list([]).
  ```

  Now add the induction step:

  ```
  list([H|T]) :- list(T).
  ```

- In English: It's a list if it's empty or if its tail is a list.

Questions?

List Processing

- Yet another list processing problem; How can Prolog prove that something is the last element of a list?

List Processing

- Yet another list processing problem; How can Prolog prove that something is the last element of a list?
- How do you prove `last([a,b,c],c)`?

List Processing

- Yet another list processing problem; How can Prolog prove that something is the last element of a list?
- How do you prove `last([a,b,c],c)`? Induction.

List Processing

- Yet another list processing problem; How can Prolog prove that something is the last element of a list?
- How do you prove `last([a,b,c],c)`? Induction.
- Prove `last([b,c],c).`

List Processing

- Yet another list processing problem; How can Prolog prove that something is the last element of a list?
- How do you prove `last([a,b,c],c)`? Induction.
- Prove `last([b,c],c).`
- Prove `last([c],c).`

List Processing

- Yet another list processing problem; How can Prolog prove that something is the last element of a list?
- How do you prove `last([a,b,c],c)`? Induction.
- Prove `last([b,c],c).`
- Prove `last([c],c).` That's easy: *The atom **`c`** is the last thing in the list if it's the only thing in the list:*

List Processing

- Yet another list processing problem; How can Prolog prove that something is the last element of a list?
- How do you prove `last([a,b,c],c)`? Induction.
- Prove `last([b,c],c).`
- Prove `last([c],c).` That's easy: *The atom* ***`c`*** *is the last thing in the list if it's the only thing in the list:*

  `last([H|[]],H).` or `last([H],H).`

List Processing

- Yet another list processing problem; How can Prolog prove that something is the last element of a list?
- How do you prove `last([a,b,c],c)`? Induction.
- Prove `last([b,c],c).`
- Prove `last([c],c).` That's easy: *The atom* ***`c`*** *is the last thing in the list if it's the only thing in the list:*

  `last([H|[]],H).` or `last([H],H).`

- What's the induction step?

List Processing

- Yet another list processing problem; How can Prolog prove that something is the last element of a list?

- How do you prove `last([a,b,c],c)`? Induction.

- Prove `last([b,c],c).`

- Prove `last([c],c).` That's easy: *The atom* ***c*** *is the last thing in the list if it's the only thing in the list:*

  `last([H|[]],H).` or `last([H],H).`

- What's the induction step?

```
last([H|T],X) :- last(T,X).
```

List Processing

- The power of recursion:
- We can query the same procedure with a variable instead of an atom and get the last element of a list.

```
?- last([a,b,c],X).

X = c.
```

List membership

What happens when:

```
?- member(b,[a,b,c]).

?- member(X,[a,b,c]).

?- member(b,X).

?- last([c|[a,b]],X).

?- last([],X).
```

```
On your own: trace the execution of these
procedures through the resolution algorithm.
```

Questions

Next Class

- We will talk more about Lists, Recursion and Resolution and see more examples.
- We have covered Chapters 6 and 7 so it's a good idea to read it now.
- Next Week: Arithmetic, Cuts and Negation (Ch. 8 & 11)