Overview Prolog lectures

- Lecture 1
  - Introduction “Family database”: first Prolog program, Prolog queries, Prolog execution tree
- Lecture 2
  - Syntax and Meaning of Prolog Programs (Chapter 2)
- Lecture 3
  - Lists, Operators, Arithmetic (Chapter 3)
- Lecture 4
  - Example Programs (from Chapter 4)
  - Trees in Prolog (from Chapter 9)

Prolog overview

- Lecture 5
  - Controlling backtracking (Chapter 5)
  - Built-in Predicates (Chapter 6)
- Lecture 6
  - More Prolog programming (from Chapters 8 and 9)

Prolog Lecture 1

Introduction "Family database": first Prolog program, Prolog queries, Prolog execution tree

Programming in Prolog

- The language is based on Horn-clause logic (logical connectors, truth tables):
  - Describe/ specify the problem domain
  - High-level specification
- Aspects of the execution by a Prolog system:
  - Logical variables and unification
  - Resolution, selection strategy and backtracking
  - Automatic garbage collection

Prolog facts

```
father(paul,koen).      mother(els,vincent).
father(paul,els).       mother(els,edith).
father(koen,eefje).     mother(els,pieter).
father(koen,marten).    mother(denise,koen).
```

represent unconditional knowledge: what is true

Reading it in natural language:

paul is the father of koen

paul is the father of els

...

denise is the mother of koen

Alternative readings possible?

/* Documentation % */

```
% father(Father, Child) is true
%  if Father is the father of Child

                % for comments till eol

/* mother(Mother, Child)is true
   if Mother is the mother of Child
*/
```

Reason about the knowledge using natural language

```
father(paul,koen).      mother(els,vincent).
father(paul,els).       mother(els,edith).
father(koen,eefje).     mother(els,pieter).
father(koen,marten).    mother(denise,koen).
```

Is koen the father of marten?

Yes

Who is the mother of edith?

els

Who has a father?

....

- Run this in SWISH

  https://swish.swi-prolog.org/p/GJ_DL_2021_Lecture1.swinb

- Run this on your PC

Same reasoning in Prolog: query

```
father(paul,koen).        mother(els,vincent).
father(paul,els).         mother(els,edith).
father(koen,eefje).       mother(els,pieter).
father(koen,marten).      mother(denise,koen).
```

Is koen the father of marten?	Yes
`?- father(koen,marten).`	`true`
Who is the mother of edith?	Els
`?- mother(X,edith).`	`X = els`
Who has a father?	....
`?- father(_,X).`	`X = koen or X = els or X = eefje or X = marten`

Put the Prolog program in a file: family.pl and start SWI-Prolog

```
Welcome to SWI-Prolog (threaded, 64 bits, version 7.4.1)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free
   software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- [family].                    % consult the family.pl file
true
?- father(koen,marten).
true.
?- mother(X,edith).
X = els.            % ; for getting the next solution
?-
```

```
...
?- father(_,X).
X = koen ;                  % ; for getting the next answer
X = els ;
X = eefje ;
X = marten.                 % the last answer!

?- father(_,X).
X = koen ;
X = els <CR>                % <CR> no more answers


% try h instead of ; or <CR> !!!
```

More complex queries related to father/2 and mother/2

- Is paul the grandfather of marten by the side of his father?
- Who is the wife of paul?
- Who is a sibling of edith?
- Is koen the father of els?

You can assume an old-fashioned family (from the previous century….).

More complex queries

- Is paul the grandfather of marten by the side of his father?
```
?- father(X,marten), father(paul,X).    % conjunction
```
- Who is the wife of paul?
```
?- father(paul,X), mother(V,X).
```
- Who is a sibling of edith?
```
... ?- mother(X,edith),mother(X,BZ).    % 3 answers
```
- Is koen the father of els?
  closed world assumption

Alternative for composed queries: grandfather relation

- What are the (logical) rules for this relation? When does the relation hold?
  - Somebody is the grandfather of a person if he is the father of yet another person that is the father that person.
  - Somebody is the grandfather of a person if he is the father of yet another person that is the mother that person
  - Thus 2 rules: Prolog rules (aka Prolog clauses)
```
grandfather(S,P) :- father(S,X), father(X,P).
grandfather(S,P) :- father(S,X), mother(X,P).
```
- Is there a **and** or a **or** between the rules?

Reading Prolog rules

```
grandfather(S,P) :- father(S,X), mother(X,P).
```

- (for all S and P it holds that) S is the grandfather of P if there exists a X such that S is the father of X and X is the mother of P.
- for all S, X and P it holds that if S is the father of X and X is the mother of P then S is the grandfather of P.
- To find out that S is the grandfather of P, you first look for a X such that S is the father of X and then you check that X is the mother of P.

Declarative and procedural readings of the clauses

Execution of Prolog rules

The execution of a Prolog program corresponds to the procedural reading

```
grandfather(S,P) :- father(S,X), father(X,P).
grandfather(S,P) :- father(S,X), mother(X,P).
```

to find out that S is the grandfather of P, you first look for a X such that S is the father of X and then you check that X is the father of P.

or (in case of failure)

you first look for a X such that S is the father of X and then you check that X is the mother of P.

Prolog query = challenge for Prolog

```
?- grandfather(koen,els).
```

Try to prove that grandfather(koen,els) is true

```
?- grandfather(X,vincent).
```

Try to find the X-s for which grandfather(X,vincent) is true

This process is known as the refutation of a negation

not(there exists a X such that grandfather(X,vincent) is true)

```
?- grandfather(X,vincent).
```

Some terminology and syntax

```
grandfather(S,P) :- father(S,X), mother(X,P).
```

`grandfather(S,P)` is the head of the clause

`father(S,X), mother(X,P)` is the body of the clause

in this case the body is a conjunction

`grandfather(S,P),father(S,X)` and `mother(X,P)` are literals

`father(S,X)` and `mother(X,P)` are goals

`grandfather/2,father/2` and `mother/2` are predicate symbols, their arity is 2, their names are grandfather, father and mother

`S,X` and `P` are variables

in `father(S,X)` are `S` and `X` the arguments of the literal

in the program we had the constants `paul, vincent, els` ...

(actually functor symbols with arity 0)

More terminology

```
grandfather(S,P) :- father(S,X), mother(X,P).
```

- `grandfather/2` expresses a relation between 2 objects
- the objects are Prolog terms
- a relation/predicate has no directionality
  (no input/output arguments)
- in a proof of a query, a variable can get only one value
- but, a query can have different proofs, and a variable can have different values in different proofs.

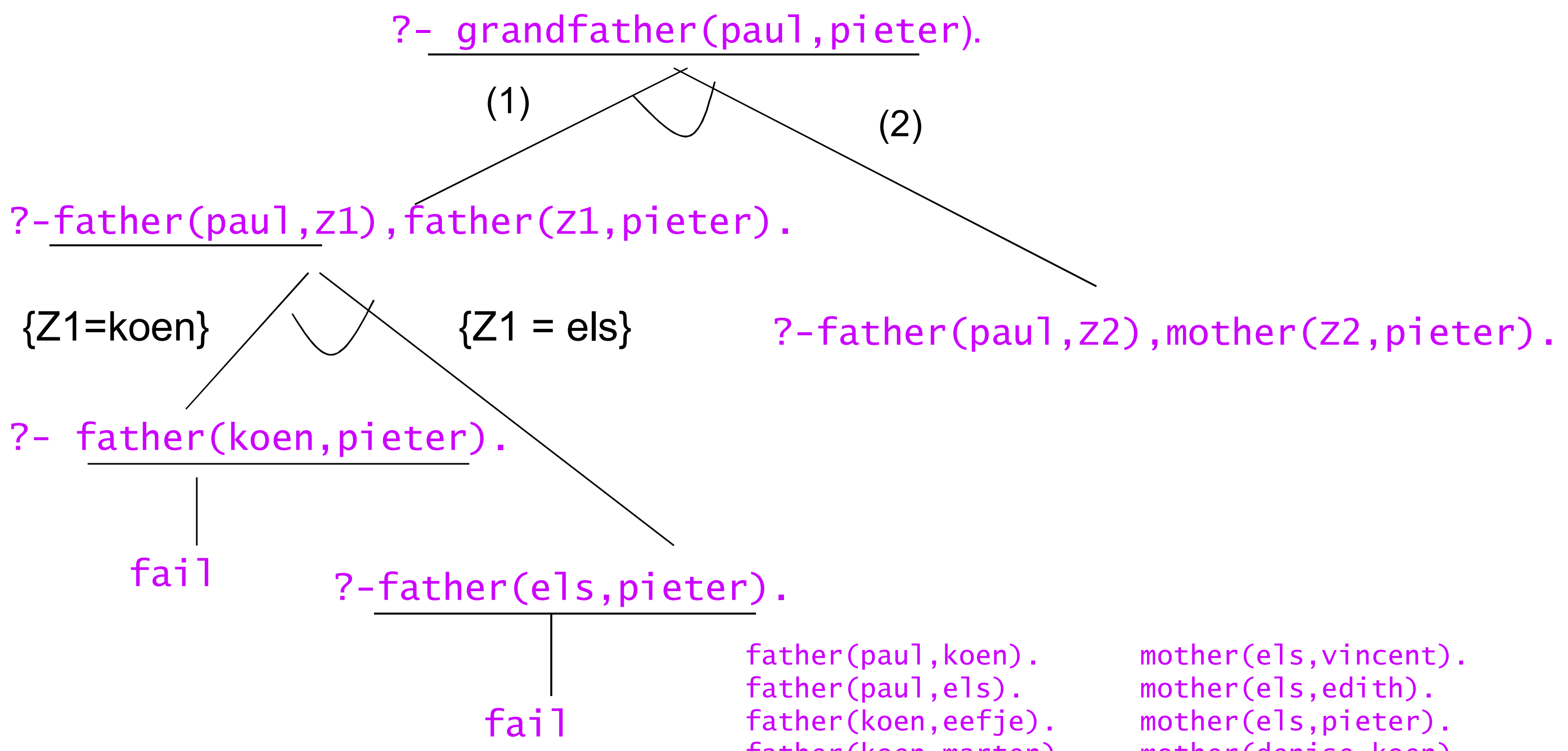
EXECUTION TREES

The scope of Prolog variables

- Variables are local to a clause
- Similar to local variables or formal parameters of a method
- Each time a clause is used (in a proof), a new set of local variables should be used.
- Can be done by renaming:

```
grandfather(S1,P1):- father(S1,X1), father(X1,P1).
grandfather(S2,P2):- father(S2,X2), father(X2,P2).
```

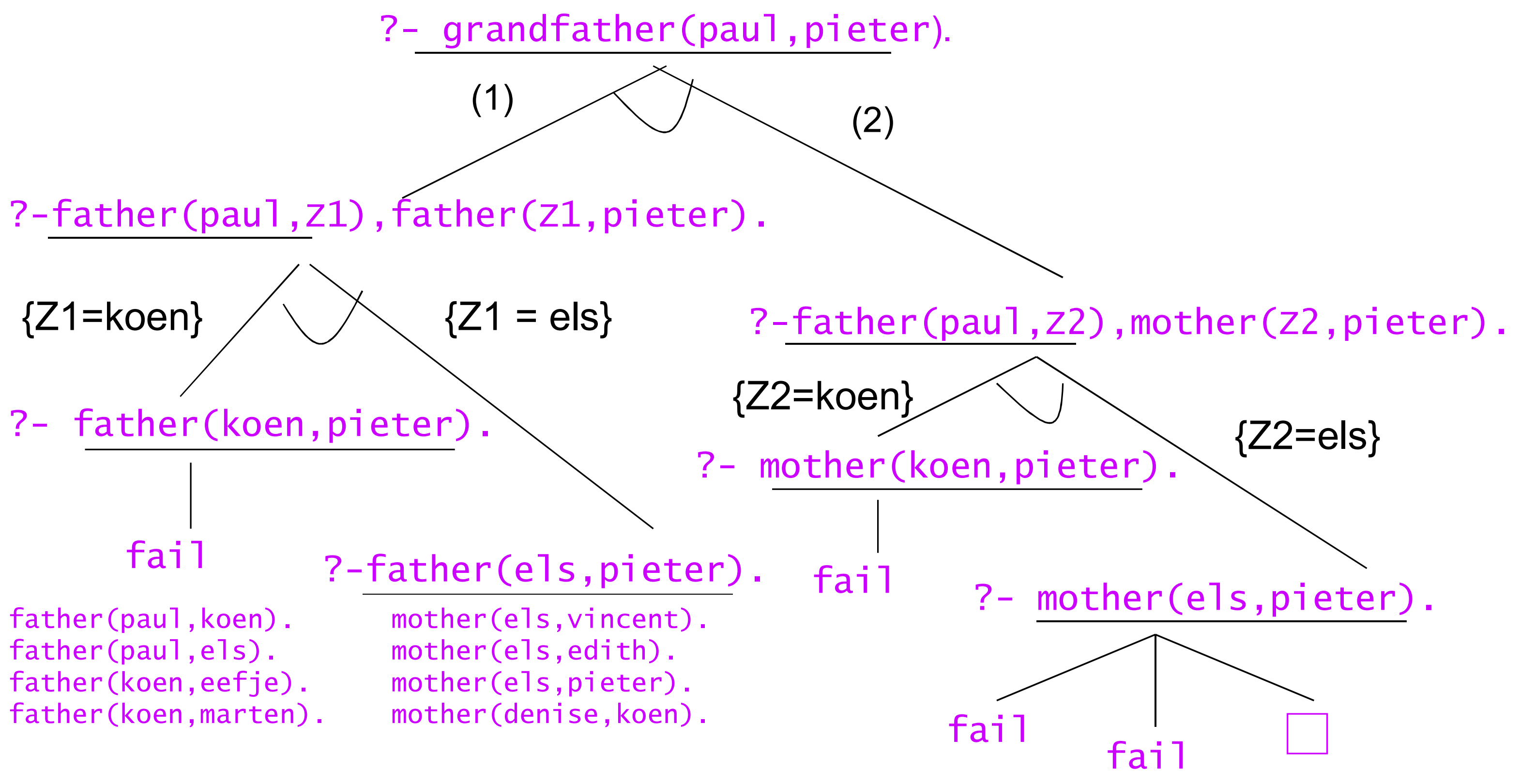
Execution tree for `?- grandfather(paul,pieter).`


?- grandfather(paul,pieter).
(1)
(2)
?-father(paul,Z1),father(Z1,pieter).
?-father(paul,Z2),mother(Z2,pieter).
{Z1=koen}
{Z1 = els}
?- father(koen,pieter).
fail
?-father(els,pieter).
fail

```
father(paul,koen).      mother(els,vincent).
father(paul,els).       mother(els,edith).
father(koen,eefje).     mother(els,pieter).
father(koen,marten).    mother(denise,koen).
```

```
(1) grandfather(X,P) :-   father(X,Z), father(Z,P).
(2) grandfather(X,P) :-   father(X,Z), mother(Z,P).
```

Execution tree for `?- grandfather(paul,pieter)`



```
father(paul,koen).       mother(els,vincent).
father(paul,els).        mother(els,edith).
father(koen,eefje).      mother(els,pieter).
father(koen,marten).     mother(denise,koen).
```

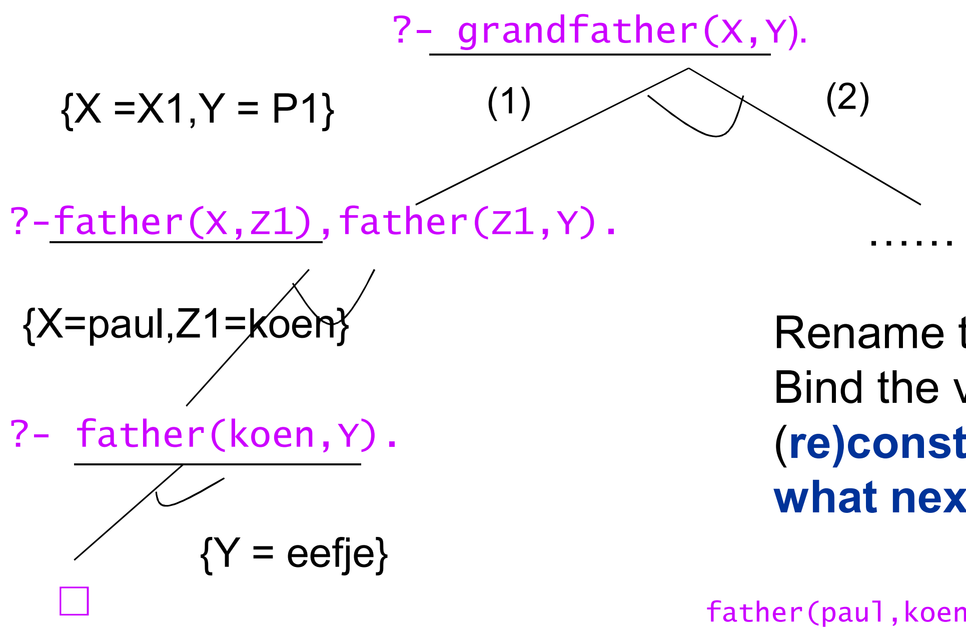
```
(1) grandfather(X,P) :-   father(X,Z), father(Z,P).
(2) grandfather(X,P) :-   father(X,Z), mother(Z,P).
```

Execution tree for `?-grandfather(paul,pieter).`

- The nodes in the tree are queries
- Outgoing edges of one node represent alternatives
- Edges have labels:
  - the selected clause
  - binding of the variable(s)
- A leaf is either fail or succes (empty query)
- Prolog goes over the tree from left to right depth-first
- This execution tree is one way to find a refutation

Swish trace of this query!

Execution tree for `?- grandfather(X,Y).`

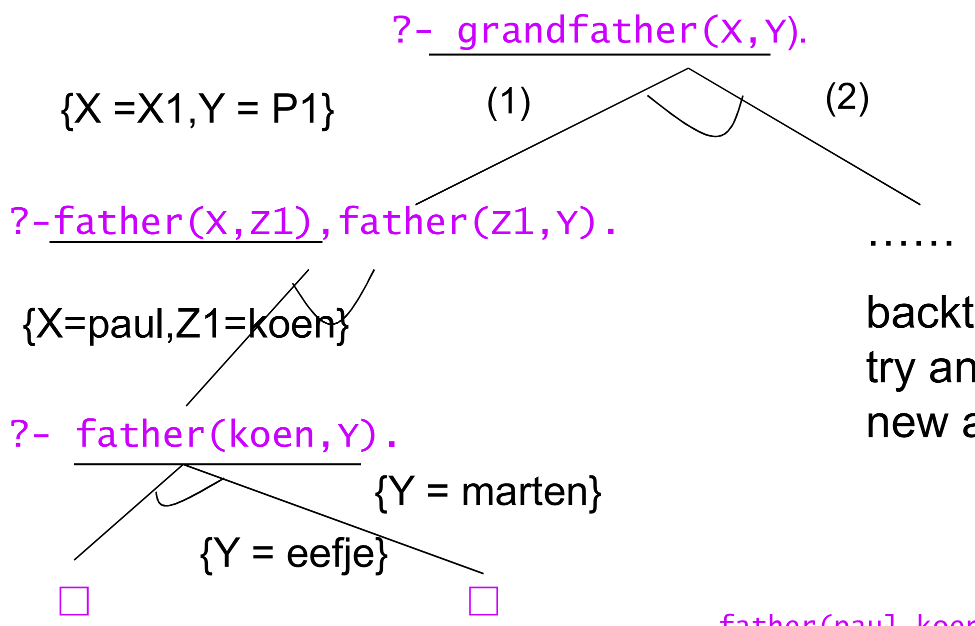


Rename the variables
Bind the variables
(re)construct the answer substitution
what next?

```
father(paul,koen).      mother(els,vincent).
father(paul,els).       mother(els,edith).
father(koen,eefje).     mother(els,pieter).
father(koen,marten).    mother(denise,koen).
```

```
(1) grandfather(X,P) :-   father(X,Z), father(Z,P).
(2) grandfather(X,P) :-   father(X,Z), mother(Z,P).
```

Execution tree for `?- grandfather(X,Y).`

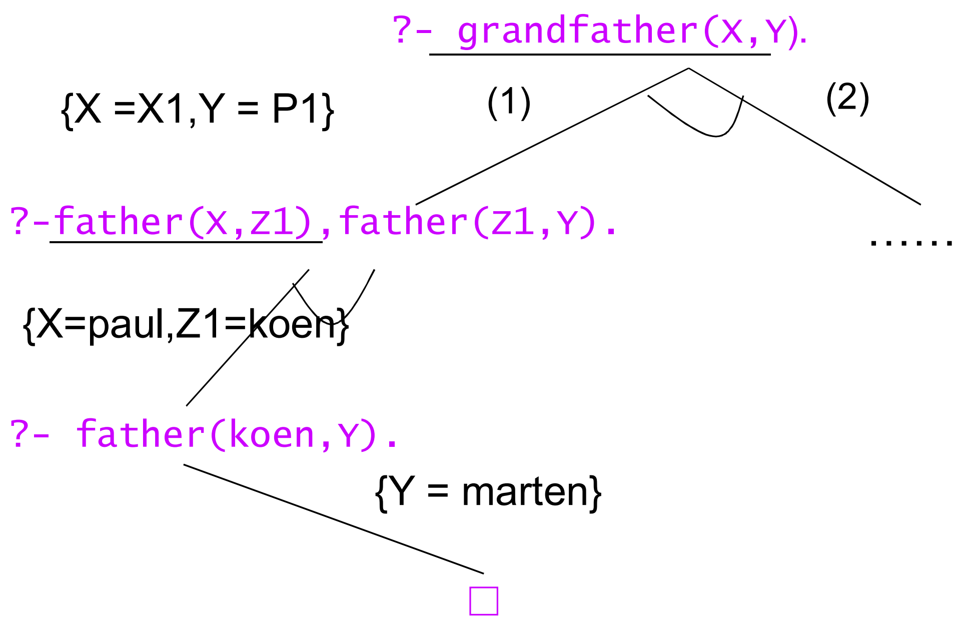


backtracking
try an alternative for ?
new answer substitution

```
father(paul,koen).      mother(els,vincent).
father(paul,els).       mother(els,edith).
father(koen,eefje).     mother(els,pieter).
father(koen,marten).    mother(denise,koen).
```

```
(1) grandfather(X,P) :-   father(X,Z), father(Z,P).
(2) grandfather(X,P) :-   father(X,Z), mother(Z,P).
```

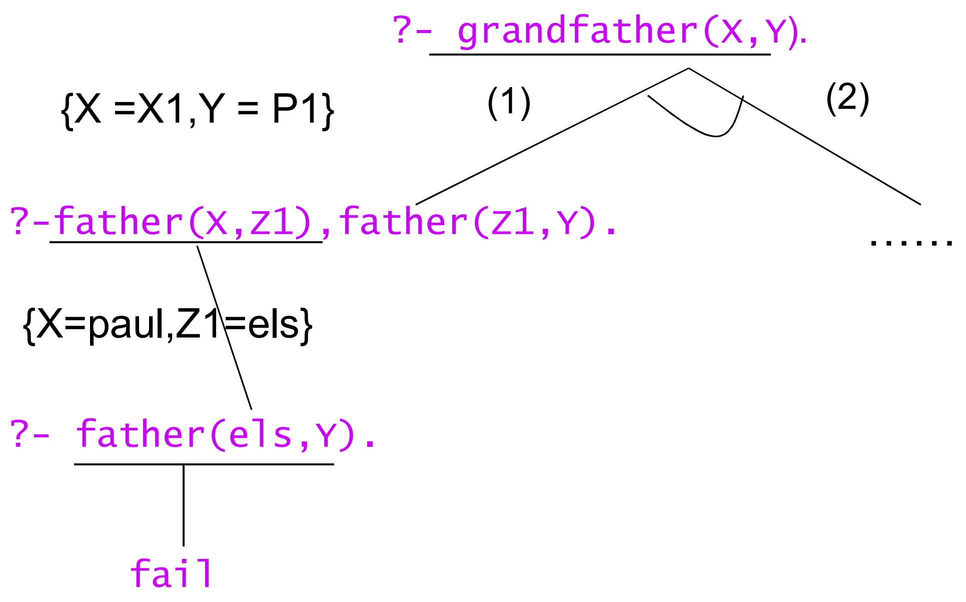
Execution tree for `?- grandfather(X,Y).`



```
father(paul,koen).      mother(els,vincent).
father(paul,els).       mother(els,edith).
father(koen,eefje).     mother(els,pieter).
father(koen,marten).    mother(denise,koen).
```

```
(1) grandfather(X,P) :-   father(X,Z), father(Z,P).
(2) grandfather(X,P) :-   father(X,Z), mother(Z,P).
```

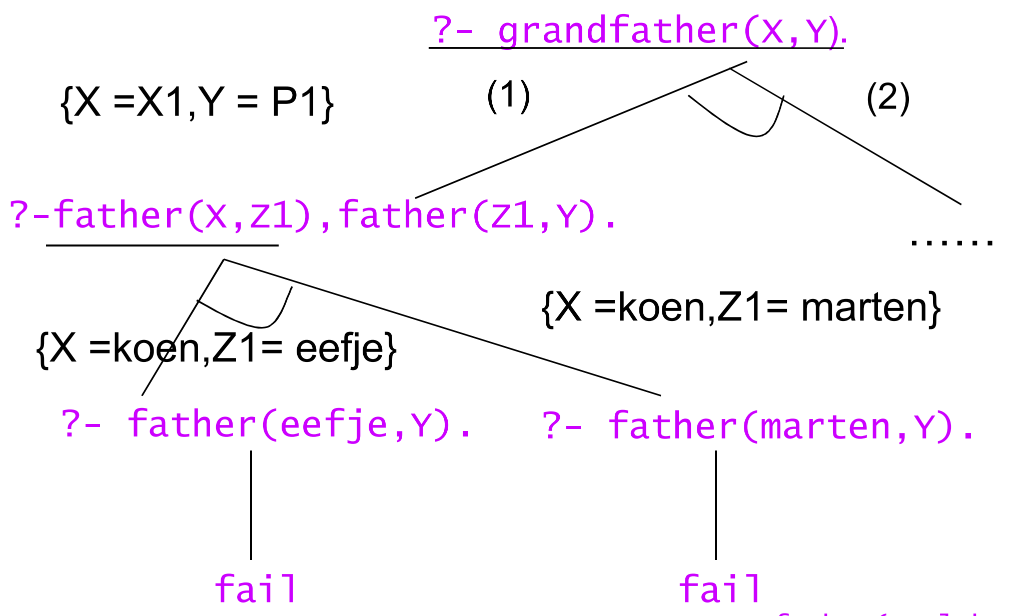
Execution tree for `?- grandfather(X,Y).`



```
father(paul,koen).      mother(els,vincent).
father(paul,els).       mother(els,edith).
father(koen,eefje).     mother(els,pieter).
father(koen,marten).    mother(denise,koen).
```

```
(1) grandfather(X,P) :-   father(X,Z), father(Z,P).
(2) grandfather(X,P) :-   father(X,Z), mother(Z,P).
```

Execution tree for ?- grandfather(X,Y).

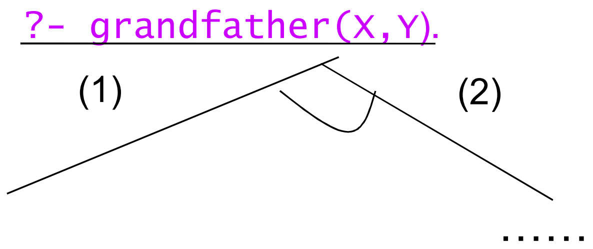


which backtrackpoint?

```
father(paul,koen).        mother(els,vincent).
father(paul,els).         mother(els,edith).
father(koen,eefje).       mother(els,pieter).
father(koen,marten).      mother(denise,koen).

(1) grandfather(X,P) :-   father(X,Z), father(Z,P).
(2) grandfather(X,P) :-   father(X,Z), mother(Z,P).
```

Execution tree for `?- grandfather(X,Y).`



Backtrack to the closest node with remaining alternatives
Forget the intermediate bindings
Try the new alternative exhaustively

```
father(paul,koen).       mother(els,vincent).
father(paul,els).        mother(els,edith).
father(koen,eefje).      mother(els,pieter).
father(koen,marten).     mother(denise,koen).
```

```
(1) grandfather(X,P) :-   father(X,Z), father(Z,P).
(2) grandfather(X,P) :-   father(X,Z), mother(Z,P).
```

Execution tree

- describes the execution ofProlog
- branches are finite or infinite
- finite branches end with fail or the empty query
- from the empty query to the root: reconstruct the answer by projecting the bindings on the variables in the query
- garbage collection for free

Resolution step = going from a node to a successor node = a logical inference step

1. Select the left most literal in the query
2. Unify the literal with the head of a renamed clause
3. Label the edge with the bindings
4. Replace the selected literal with the body of the clause
5. Apply the bindings

Resolution is based on modus tollens

```
q <- p        ~q          walk <- sunny   ~walk
__________________        _______________________
       ~p                        ~sunny
```

```
a :- d,e.     ?- a,b,c.
_______________________
    ?- d,e,b,c.
```

We bind variables and constants

- Binding 2 variables: renaming, always succeeds
- Binding a variable and a constant: always succeeds
- Binding 2 constants: succeeds for two identical constants, fails otherwise.

Is this already Prolog?

- Only variables and constants as arguments: this subset of Prolog is known as Datalog
- Datalog is a query language for databases (1978 workshop on *Logic and Databases*, Herve Gallaire and Jack Minker).
- Equivalent with propositional logic, relational databases, …
- Decidable, not Turing-complete (i.e. we are not able to count)
- Thus: adding new additional datastructures will have serious consequences