Tabled Resolution

CSE 505 – Computing with Logic

Stony Brook University

http://www.cs.stonybrook.edu/~cse505

Recap: OLD Resolution

- Prolog follows *OLD resolution*, which is SLD (*Selective Linear Definite*) resolution, but with with left-to-right literal selection & Prolog also uses the order in which the clauses are enumerated in the database to determine the order in which the branches of the search space are investigated
  - Consider a goal $\mathbf{G_1}$, $\mathbf{G_2}$, … , $\mathbf{G_n}$ as a "***procedure stack***" with $\mathbf{G_1}$, the selected literal on top
  - Call $\mathbf{G_1}$
  - **<u>If</u>** and **<u>when</u>** $\mathbf{G_1}$ returns, continue with the rest of the computation: call $\mathbf{G_2}$, and upon its return call $\mathbf{G_3}$, etc. until nothing is left
    - Note: $\mathbf{G_2}$ is "***opened up***" only when $\mathbf{G_1}$ returns, not after executing only some part of $\mathbf{G_1}$

OLD Resolution

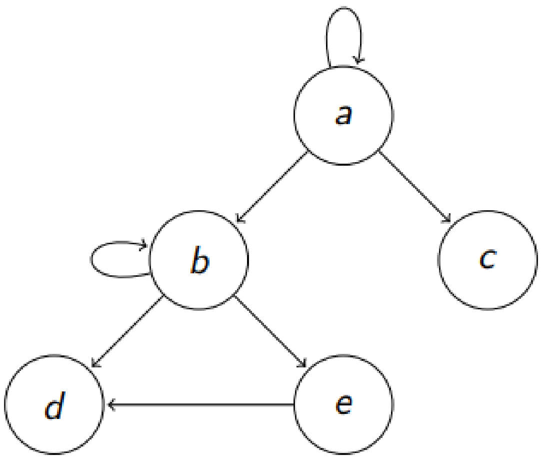
- Depth-first expansion, however, contributes to the *incompleteness* of Prolog's evaluation, which may **not terminate** even when the least model is finite (see the next example!)

Example:Reachability in Directed Graphs

- Determining whether there is a path between two vertices in a directed graph is an important and widespread problem
- For instance, consider checking whether (or not) a program accesses a shared resource before obtaining a lock:
  - A program itself can be considered as a graph with vertices representing program states!
    - A state may be characterized by the program counter value, and values of variables
    - There are richer models for representing program evaluation, but a directed graph is most basic
  - If we can go from state **`s`** by executing one instruction to **`s'`**, then we can place an edge from **`s`** to **`s'`**
  - The ***reachability question*** may be whether we can reach from the start state to a state accessing a shared resource, without going through a state that obtained a lock

Graph Reachability as a Logic Program

- A finite directed graph can be represented by a set of binary facts representing an "***edge***" relation
  - Predicate "**`q`**" on right is an example
- Reachability can then be written as a "***transitive closure***" over the edge relation
  - Observe the predicate "**`r`**" defined on right using two clauses:
    - The first clause: there is a *path* from **X** to **Y** if there is an edge from **X** to **Y**
    - The second clause: there is a *path* from **X** to **Y** if there an intermediate vertex **Z** such that:
      - there is an edge from **X** to **Z**, and
      - there is a path from **Z** to **Y**




```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

Bottom-Up Evaluation

- Note that the program on the right is a ***Datalog*** program, i.e., no function symbols
  - Its Herbrand Universe is **finite**, and its least model computation using the bottom up evaluation will **terminate:**

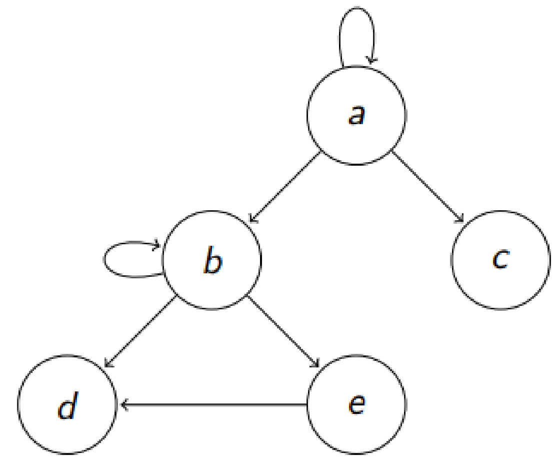
$M_0 = \emptyset$

$M_1 = T_P(M_0) = M_0 \cup \{\mathtt{q(a,a)}, \mathtt{q(a,b)}, \mathtt{q(a,c)}, \mathtt{q(b,b)}, \mathtt{q(b,d)}, \mathtt{q(b,e)}, \mathtt{q(e,d)}\}$

$M_2 = T_P(M_1) = M_1 \cup \{\mathtt{r(a,a)}, \mathtt{r(a,b)}, \mathtt{r(a,c)}, \mathtt{r(b,b)}, \mathtt{r(b,d)}, \mathtt{r(b,e)}, \mathtt{r(e,d)}\}$

$M_3 = T_P(M_2) = M_2 \cup \{\mathtt{r(a,d)}, \mathtt{r(a,e)}\}$

$M_4 = T_P(M_3) = M_3 = M_P$

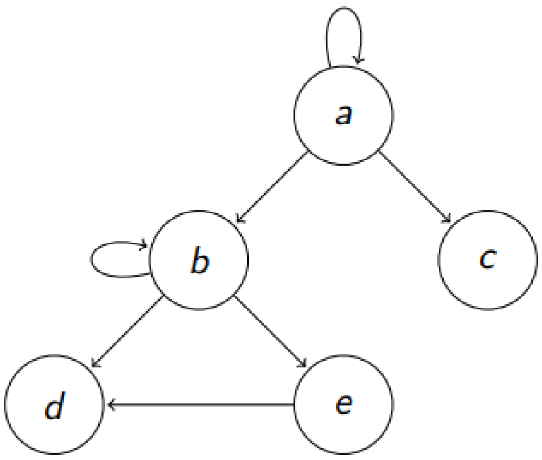



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

Bottom-Up Evaluation

M_P = {q(a,a), q(a,b), q(a,c),
q(b,b), q(b,d), q(b,e),
q(e,d), r(a,a), r(a,b),
r(a,c), r(b,b), r(b,d),
r(b,e), r(e,d), r(a,d),
r(a,e)}

- With care, using bottom-up evaluation all-pairs reachability can be computed in **O(V ·E)** time for a graph with **V** vertices and **E** edges

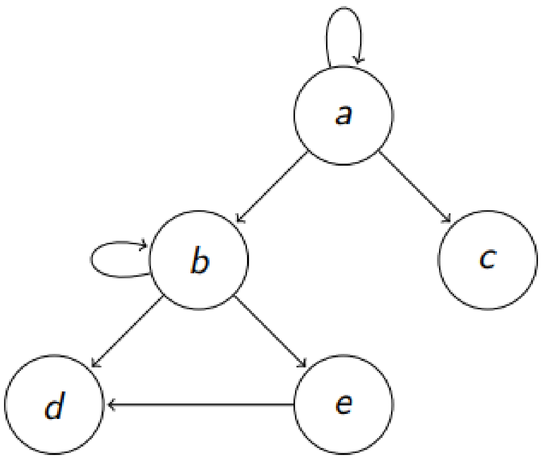



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLD Resolution with Depth-First Expansion

```
?- r(a,N).
```

Consider initial query "**`?- r(a,N).`**"
Let's construct the **SLD tree** for this query



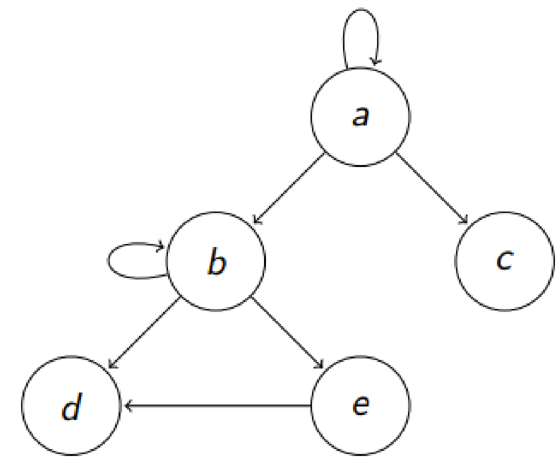

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLD Resolution with Depth-First Expansion

```
?- r(a,N).
```

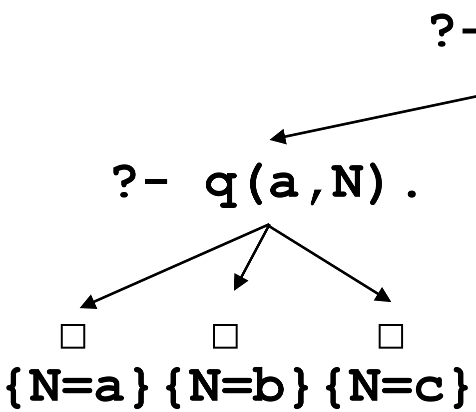
```
?- q(a,N).
```

Resolving this goal with the first clause of **`r`**, we get a new goal "**`?- q(a,N).`**"

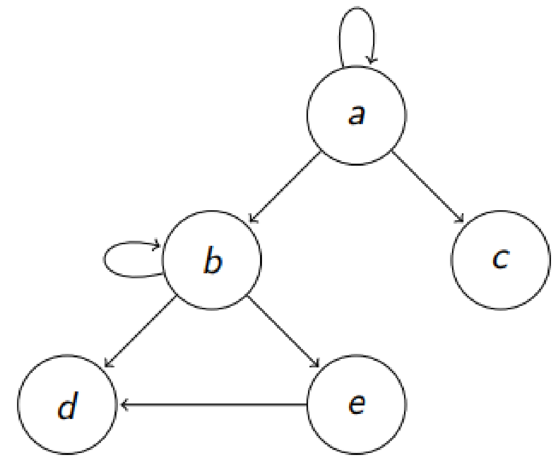



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLD Resolution with Depth-First Expansion

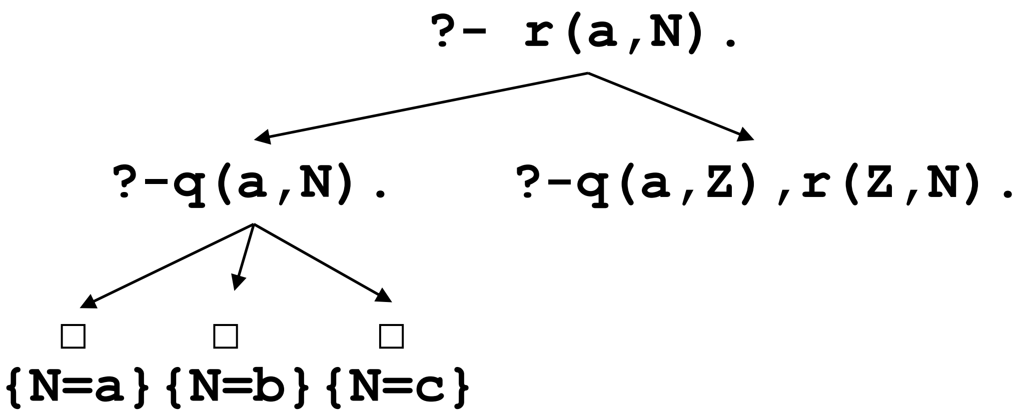


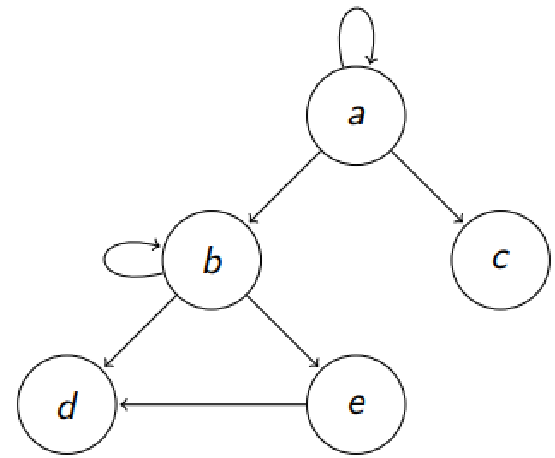
Resolving "**?- q(a,N).**" results in the empty goal, under three answer substitutions: **a**, **b**, and **c**. There are no more ways to resolve "**?- q(a,N).**"




```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLD Resolution with Depth-First Expansion



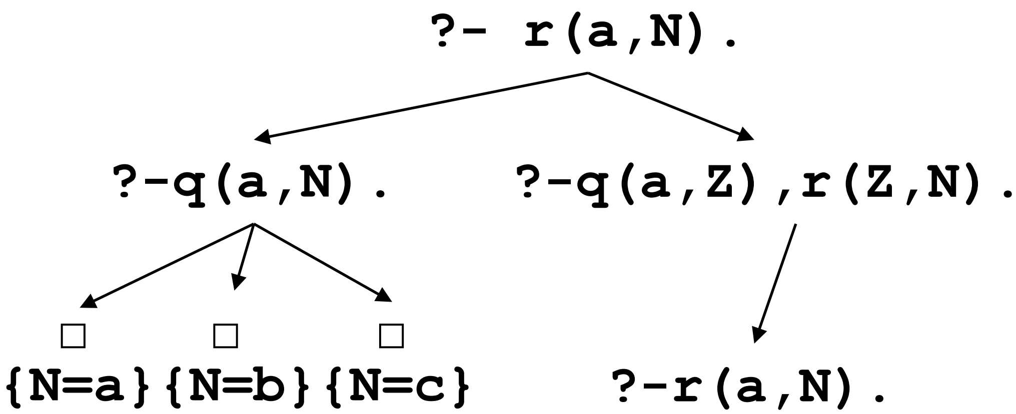


Resolving "**`?- r(a,N).`**" with the second clause of **`r`** results in goal "**`?- q(a,Z), r(Z,N).`**"

Note: We will use same variable names as in the program clause when possible, instead of mechanically inventing new variable names in every step.

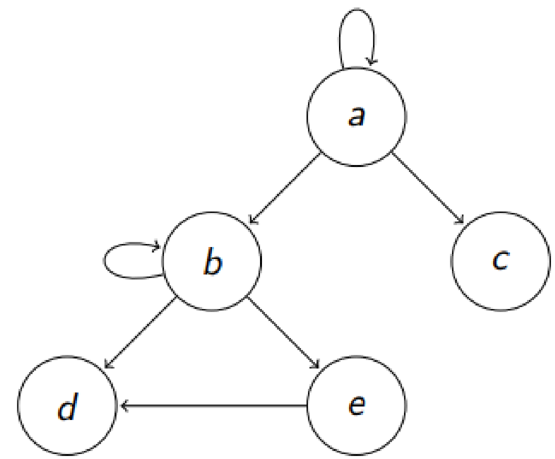
```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLD Resolution with Depth-First Expansion



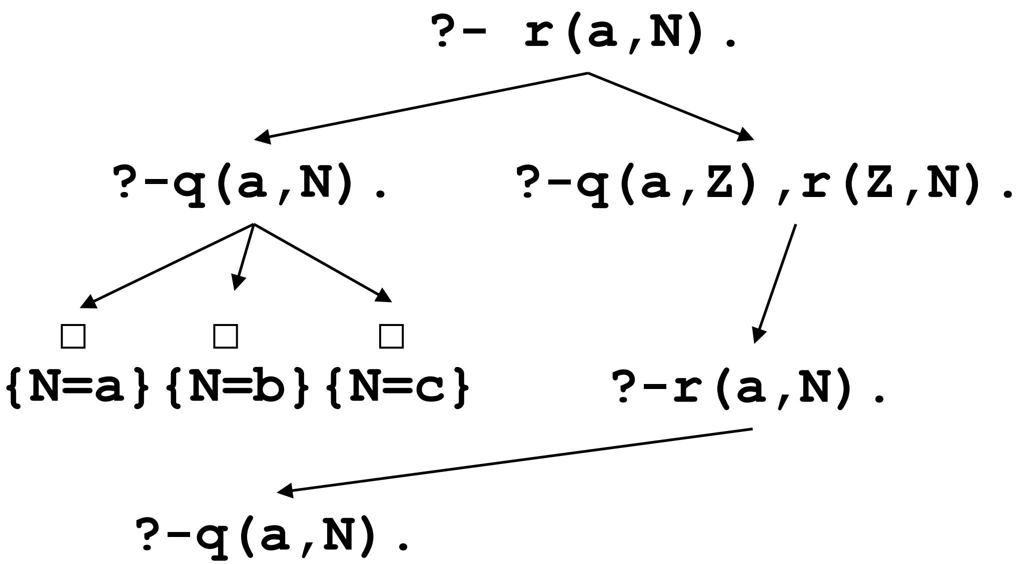
The selected literal is **`q(a,Z)`** which unifies with fact **`q(a,a)`** with **`Z=a`**. Thus we get the goal "**`?-r(a,N).`**"

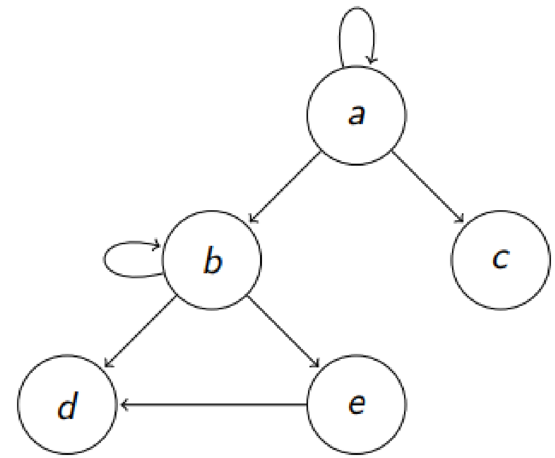
Note: This is the same goal that we had at the beginning.




```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLD Resolution with Depth-First Expansion

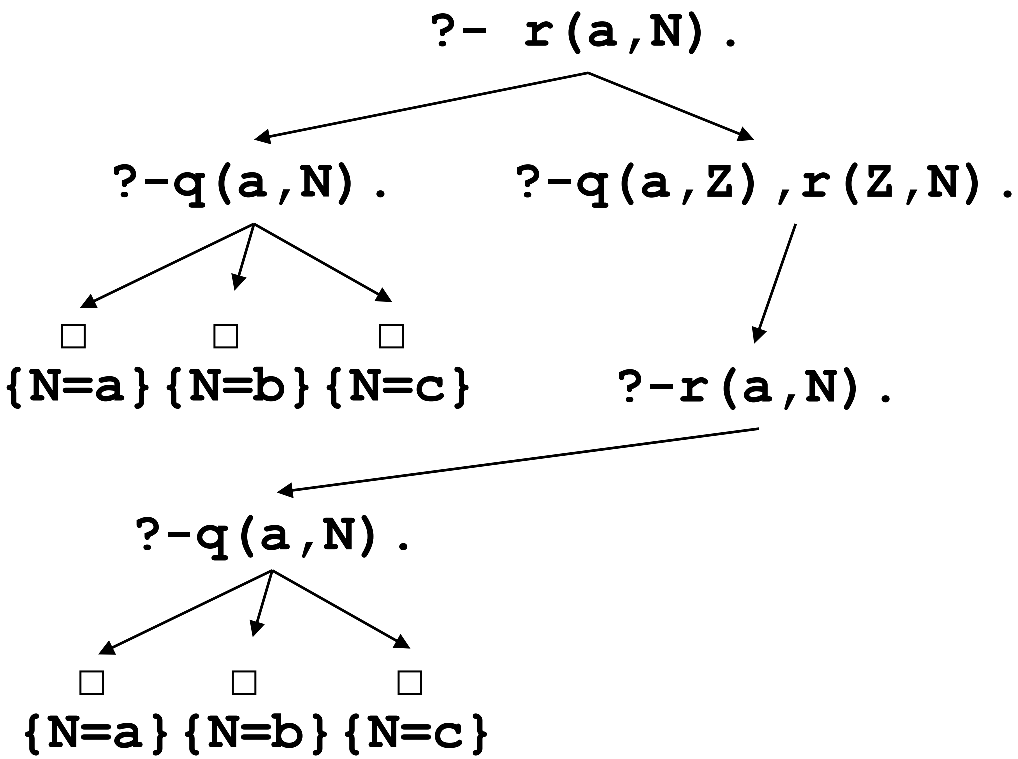


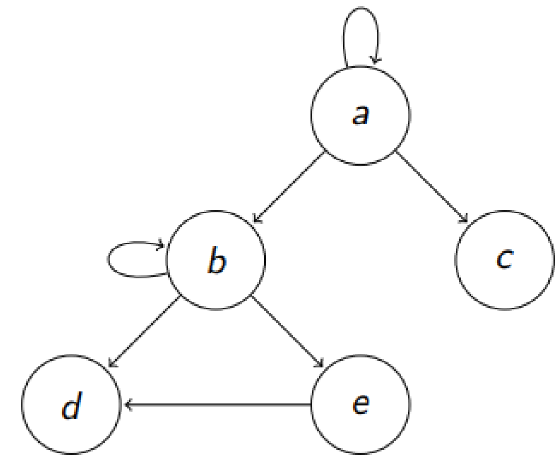



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

Resolving "**?-r(a,N).**" leads to "**?-q(a,N).**" (one of two options).

OLD Resolution with Depth-First Expansion






```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

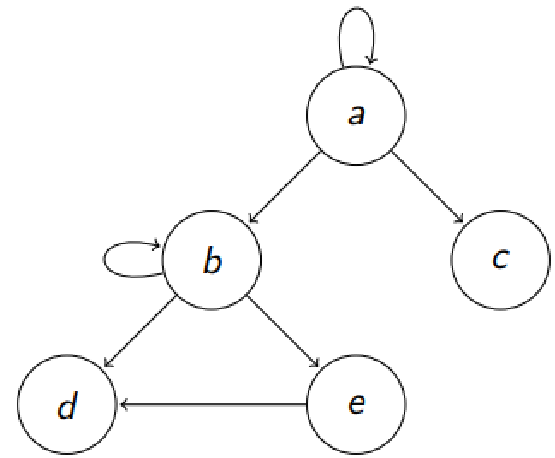
"**?-q(a,N).**", in turn, leads to empty goal with answers **N=a**, **b**, and **c.**

OLD Resolution with Depth-First Expansion

```
?- r(a,N).

?-q(a,N).          ?-q(a,Z),r(Z,N).

□      □      □
{N=a}  {N=b}  {N=c}
                   ?-r(a,N).

?-q(a,N).          ?-q(a,Z),r(Z,N).

□      □      □
{N=a}  {N=b}  {N=c}
```

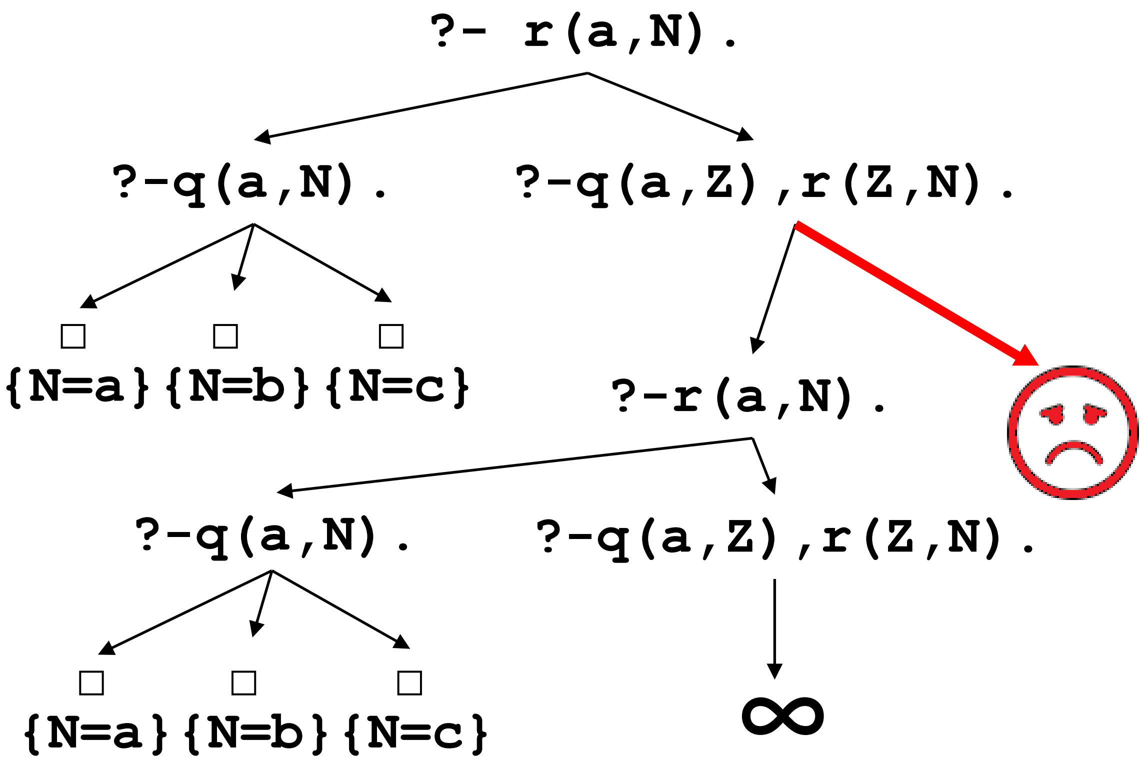
The other way to resolve "**?-r(a,N).**" is …

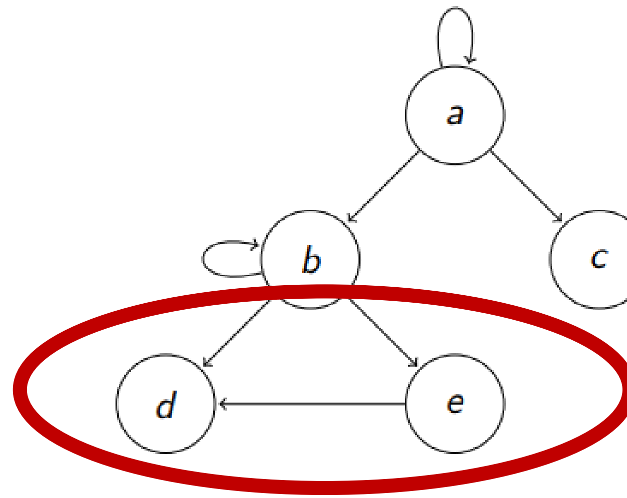
You get the drift: we are repeating work done before, and there is an infinite branch here.




```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLD Resolution with Depth-First Expansion






```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

We never get to visit the other ways of resolving "**?-q(a,Z),r(Z,N).**"

Depth-First Expansion of the OLD tree

1. If the underlying graph is acyclic, all branches in the OLD tree will be finite
2. **If the graph is cyclic, nothing to the right of an infinite branch is expanded**
   - This renders the evaluation **incomplete**: goals for which there are OLD derivations, but they are not found

- Moreover, in both cases, the same answer may be returned multiple times (even **infinitely**!)
  - Even if the underlying graph is acyclic, this evaluation is not efficient
    - For our example, for the query **`"?- r(a,N)."`** we will return **`N=b`** for each path from "**`a`**" to "**`b`**"

Expansion of the OLD tree

- **Breadth-First** expansion does have the completeness property for definite programs:
  - Every OLD derivation will be eventually constructed
    - If something is a logical consequence, we will eventually confirm it in a **finite** number of levels
  - But we may not be able to conclude ***negative information***
    - If something is not a logical consequence, we may never be able to identify it because we don't know when to stop?

Expansion of the OLD tree

- Moreover, Breadth-First expansion does not give a natural operational understanding:
  - If we view predicates as being defined by "***procedures***", then breadth first expansion **switches contexts** at the end of each step
  - As in procedural programming, **context switching is expensive** (in this case, we've to **switch substitutions**)

Programming our way around the problem

- Check for any vertex if it was not already seen in the path **L**
  - Start from **L=[]** to look for reachable vertices

```
r(X, Y) :-
        p([], X, Y).
p(L, X, Y) :-
        q(X, Y),
        not member(Y, L).
p(L, X, Y) :-
        q(X, Z),
        not member(Z, L),
        p([Z|L], Z, Y).
```

  - In **L** we remember the path so far, and use this to avoid loops

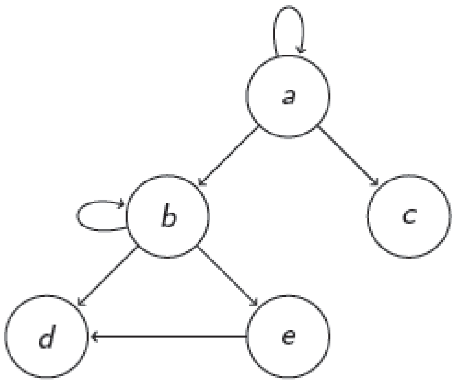
Programming our way around the problem

- We are assured termination for reachability queries
  - We stop if a node has been seen before on the same branch
- Still, this is inefficient
  - We re-execute queries on different branches of the SLD/OLD tree
  - May take exponential time

What is *Tabled Resolution*?

- ***Memoize*** calls and results to avoid repeated subcomputations
- Properties:
  - Termination: Avoid performing computations that repeat infinitely often
    - Complete for Datalog programs
  - Efficiency: Dynamically share common subexpressions

Depth-First Expansion of OLD tree with a little twist: **Stop if the entire goal has been seen before**

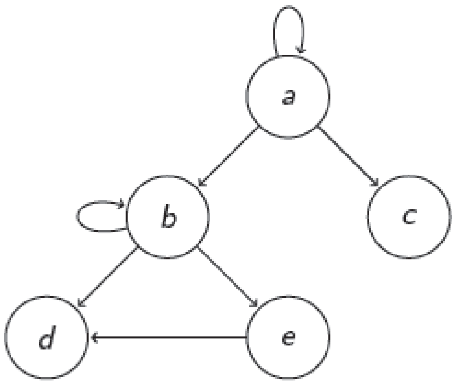


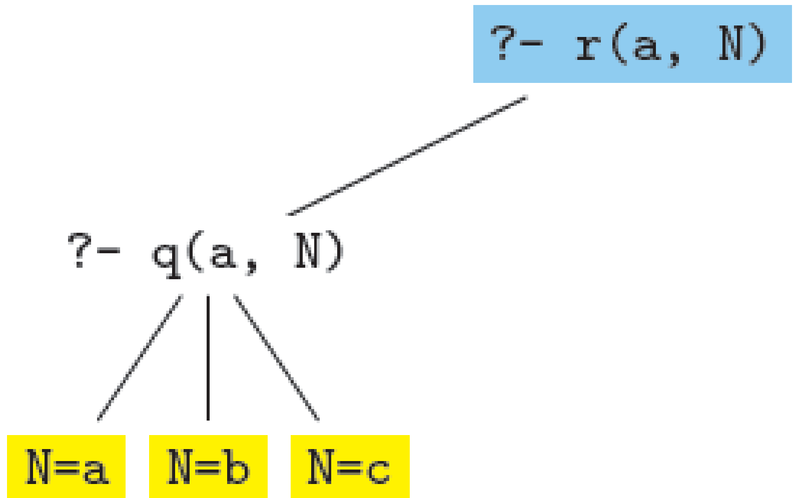

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).


r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

```
?- r(a, N)
```

the entire goal will be *tabled* (remembered)

Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before

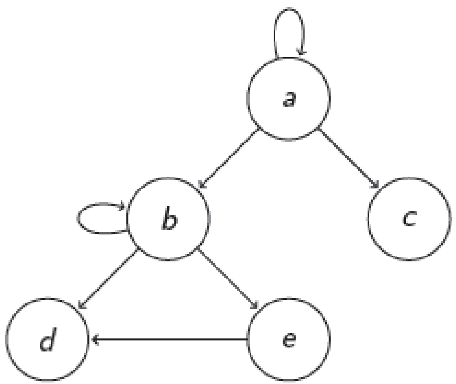


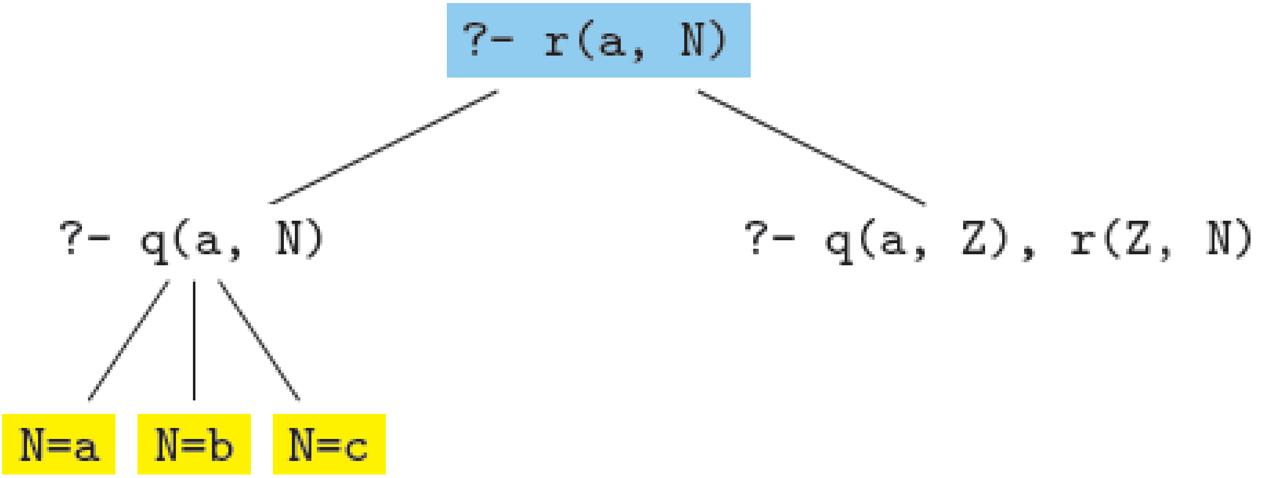



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

```
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

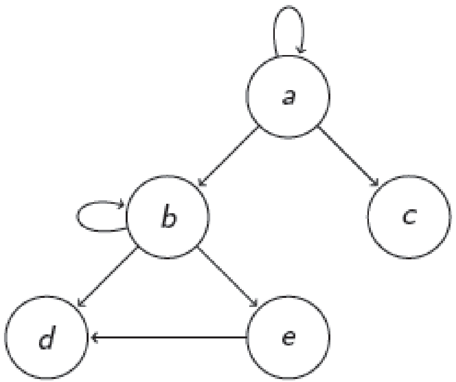
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.

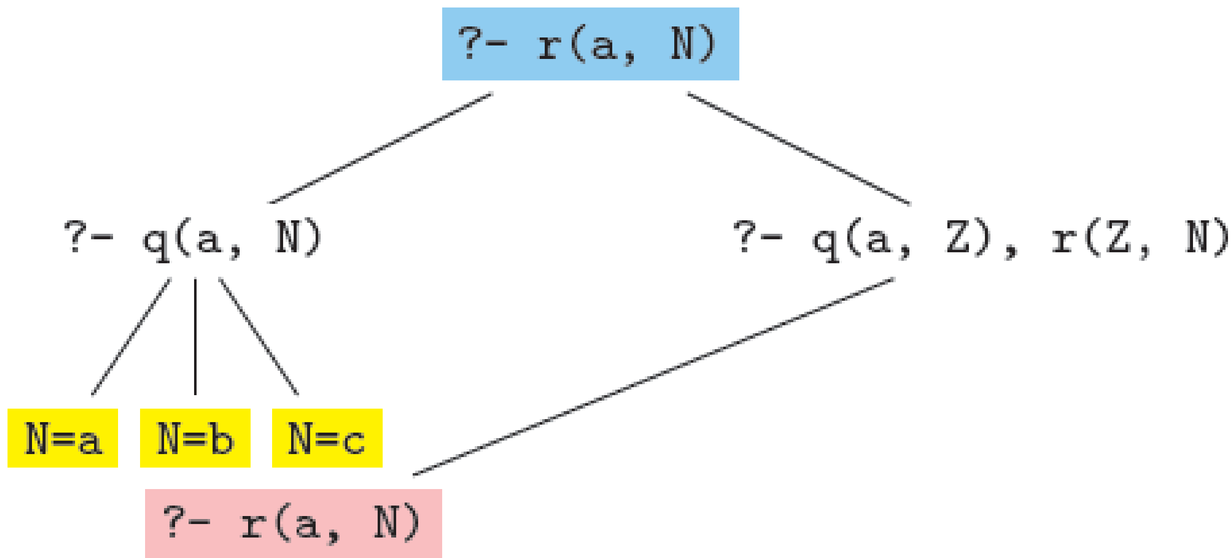





```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).


r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

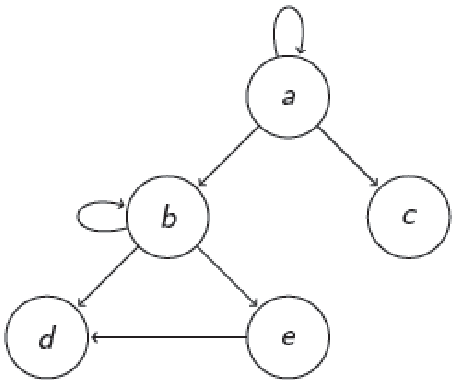
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.



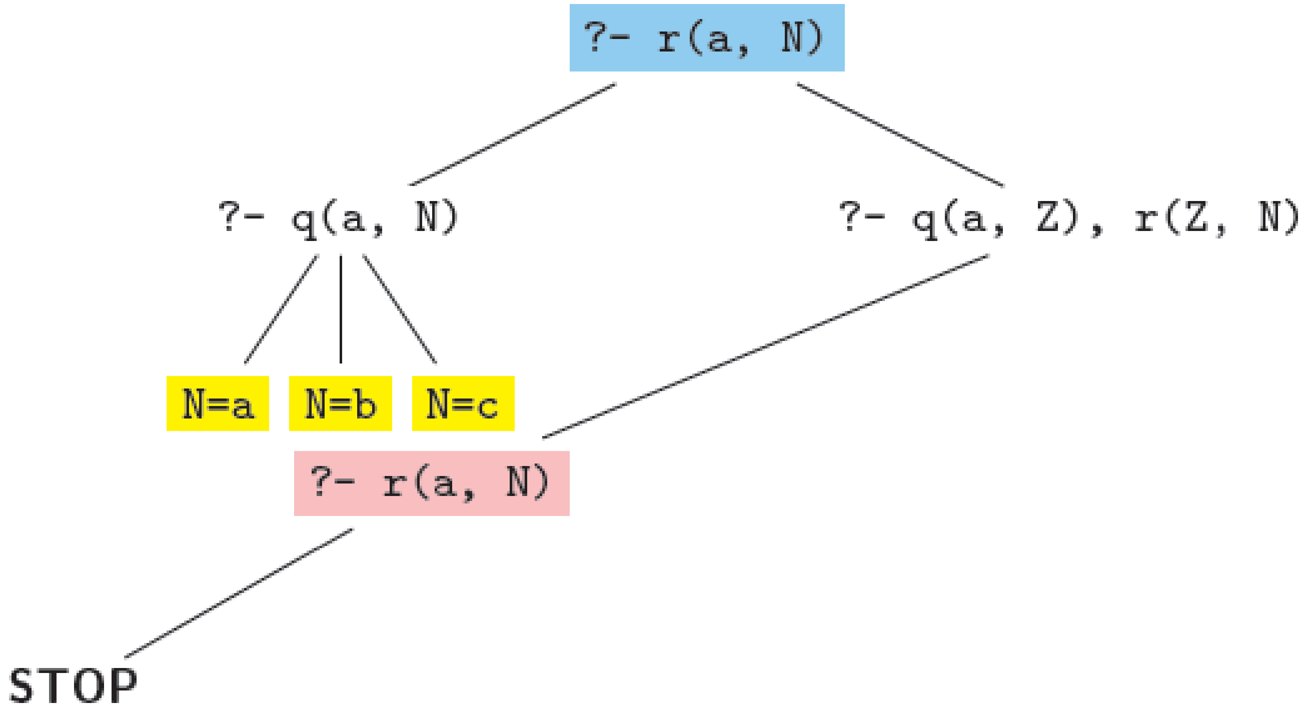



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).


r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

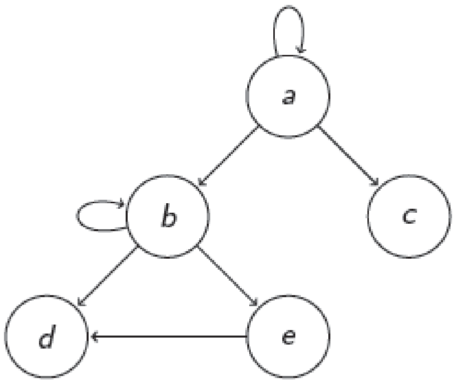
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.



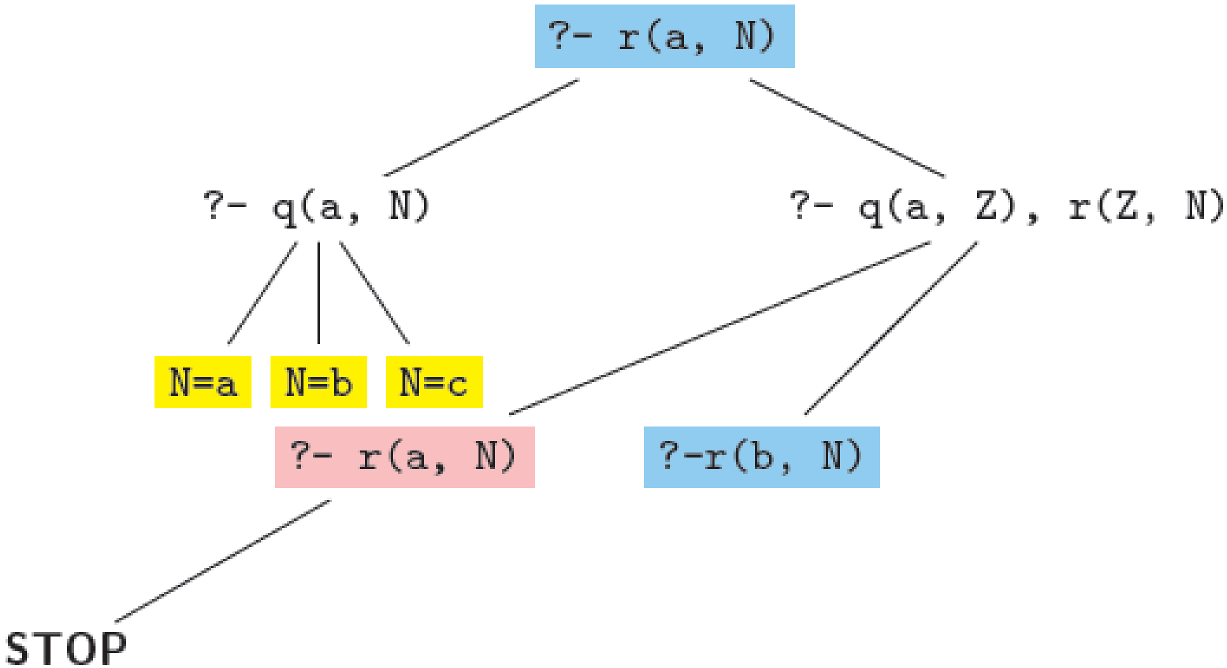

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).


r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```



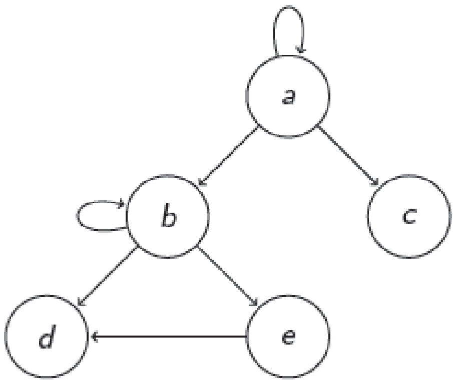
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.



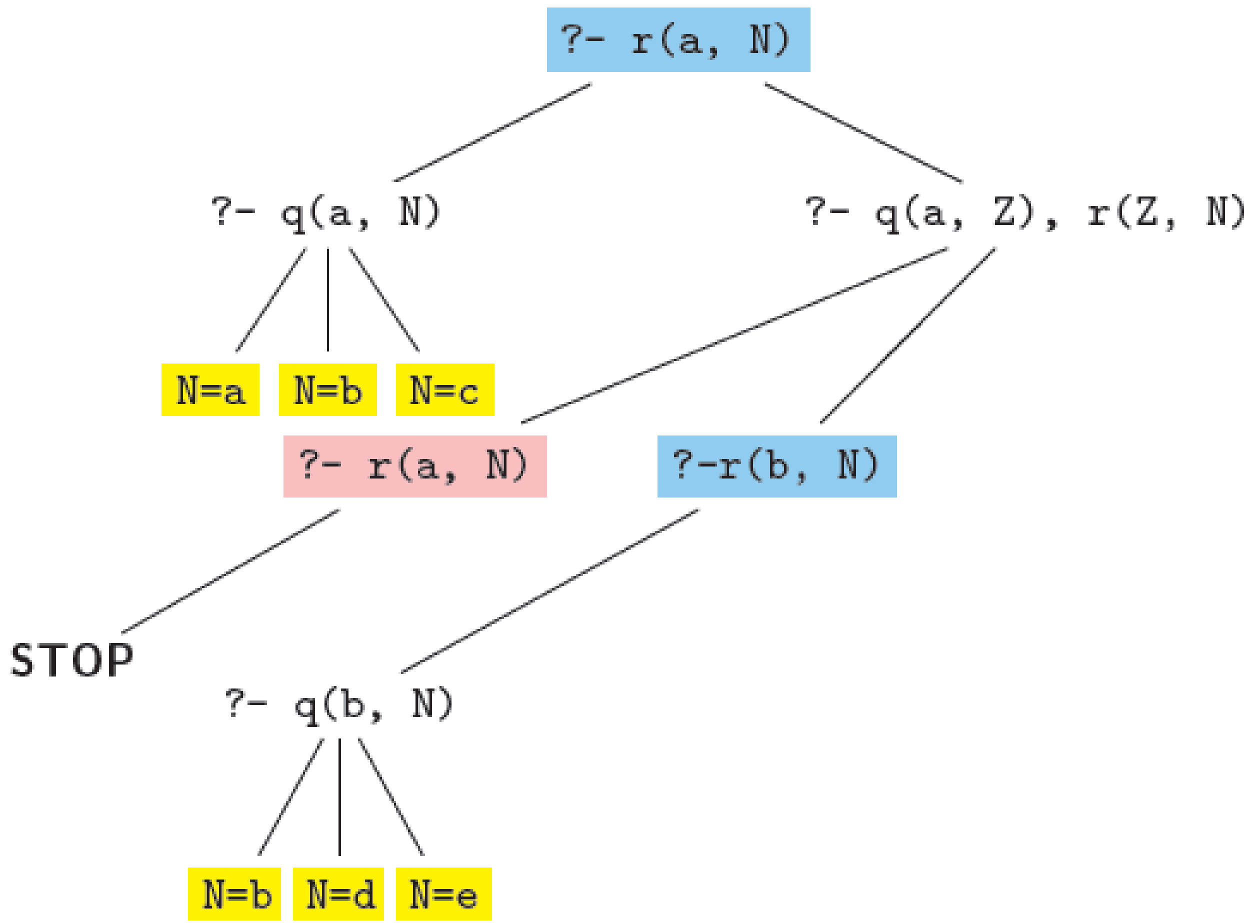

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).


r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```



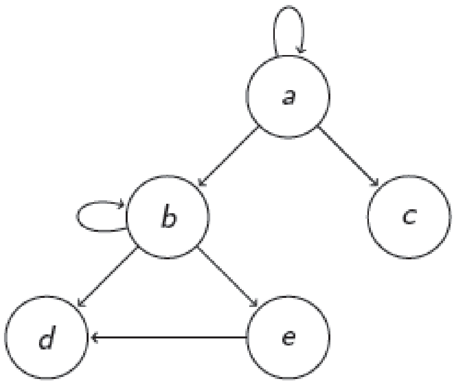
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.



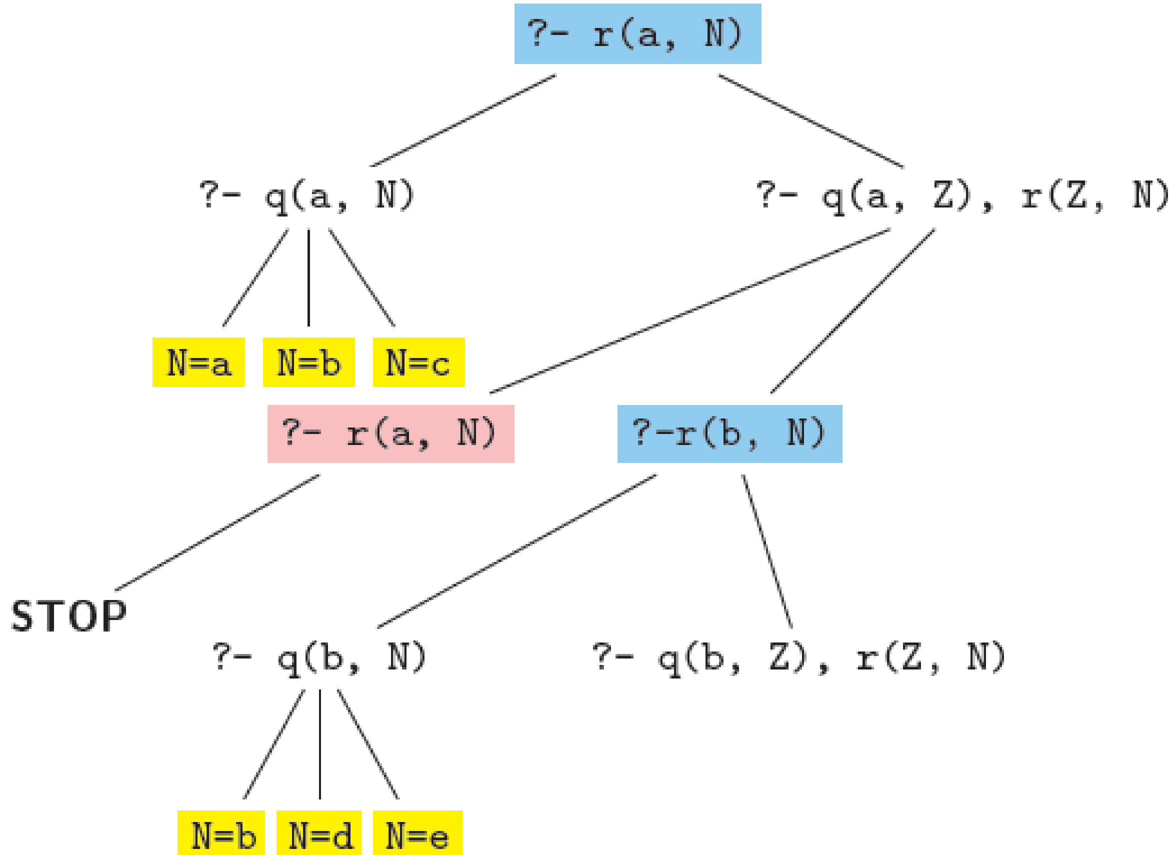

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).


r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```



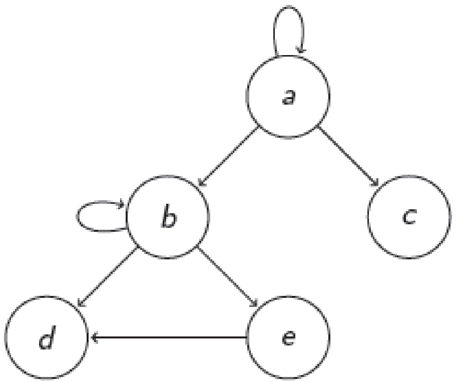
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.



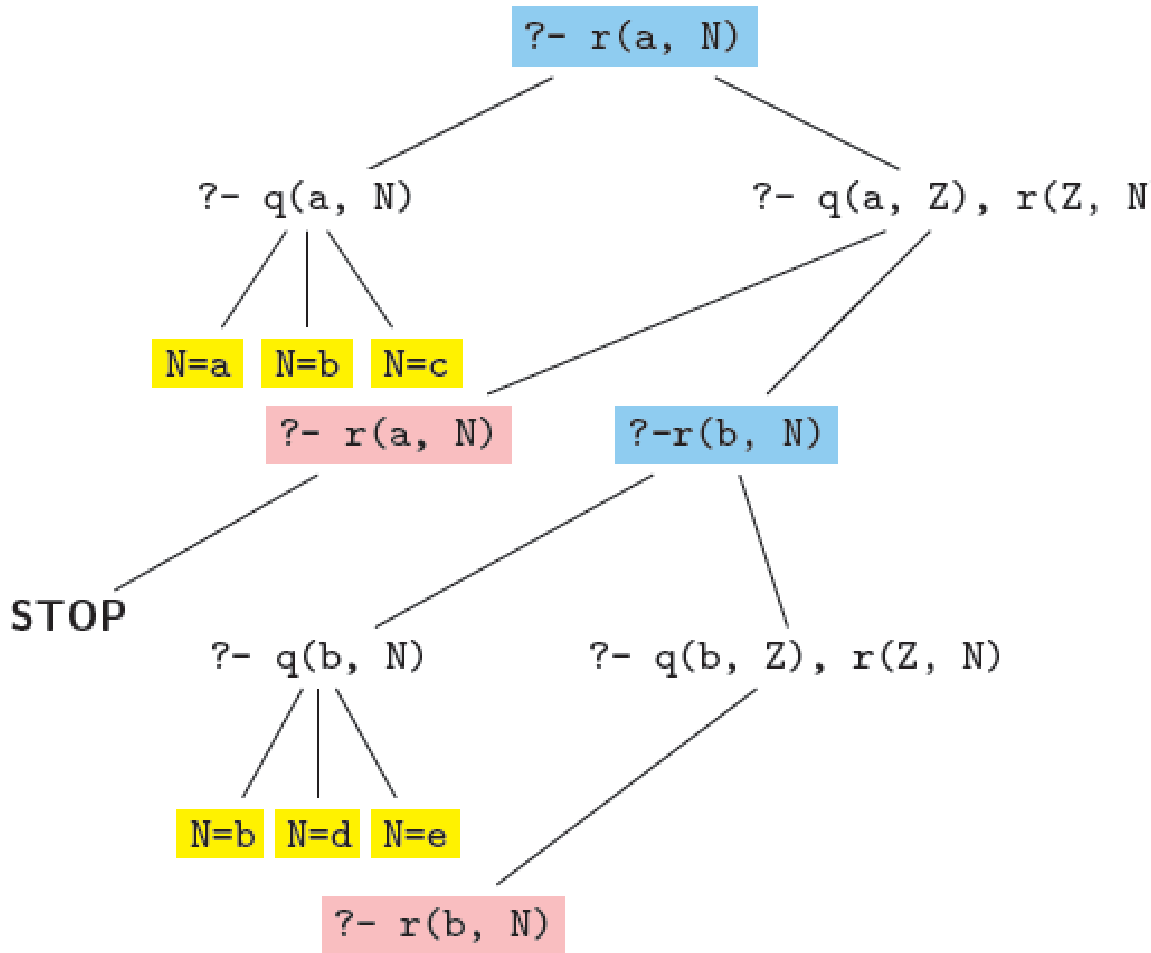

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).


r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```



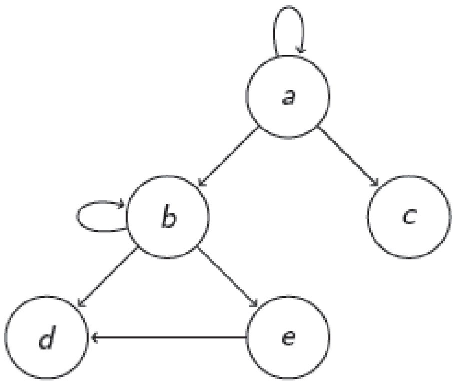
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.



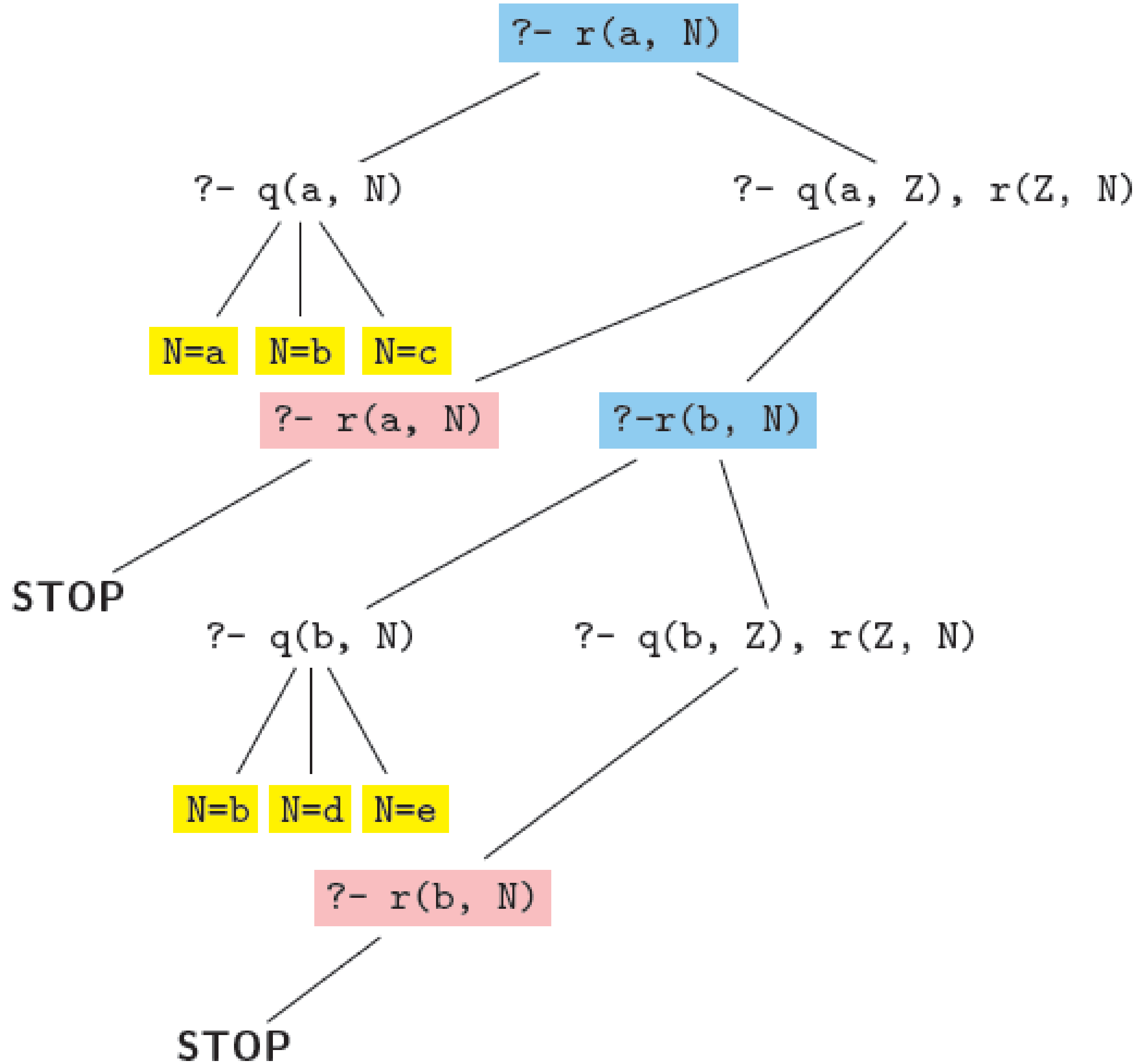

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).


r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```



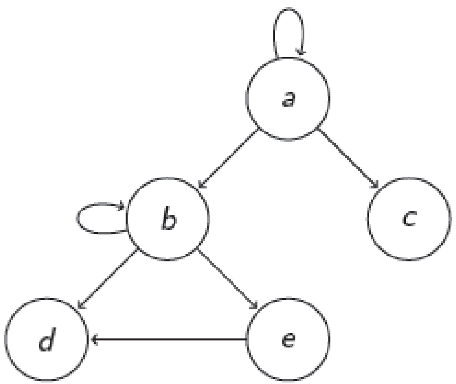
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.



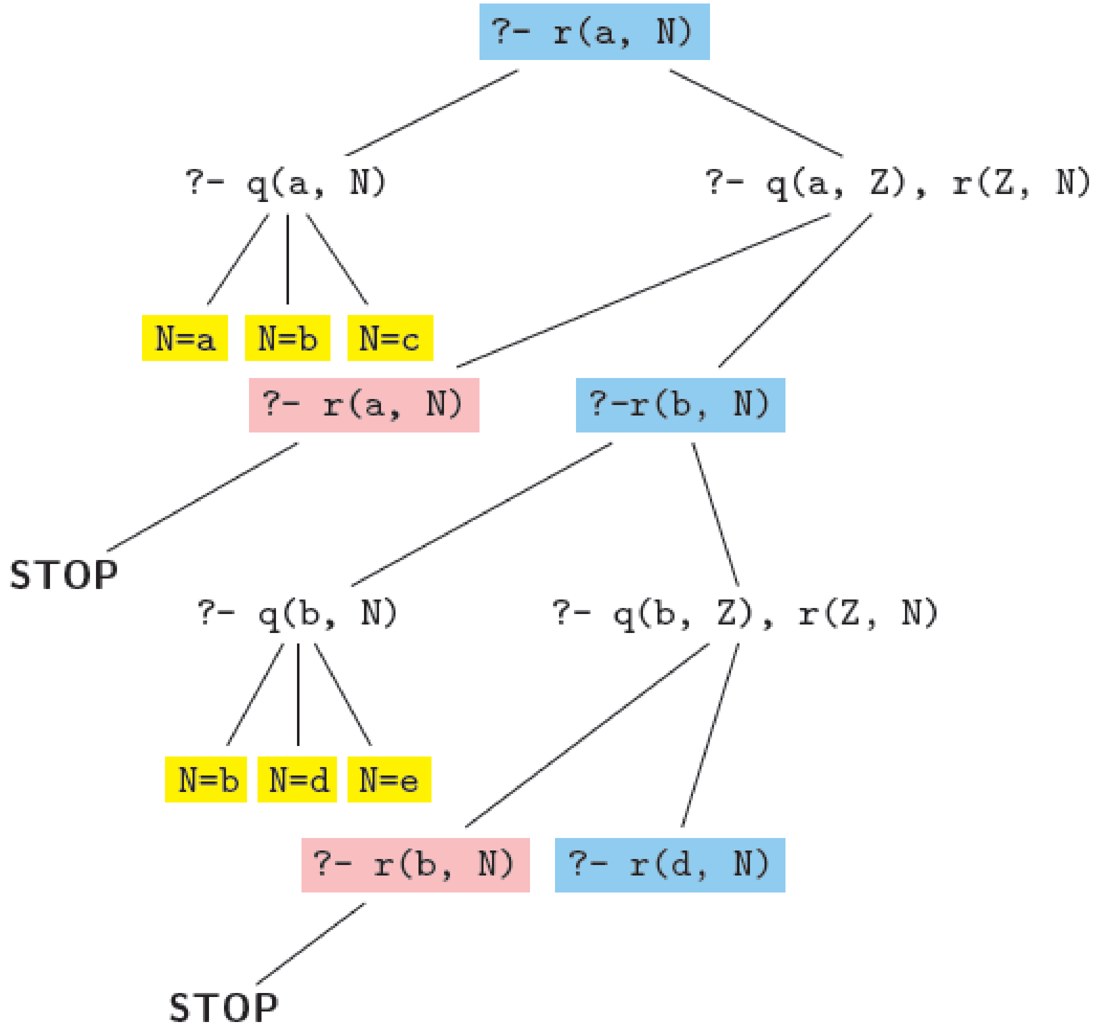

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).


r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```



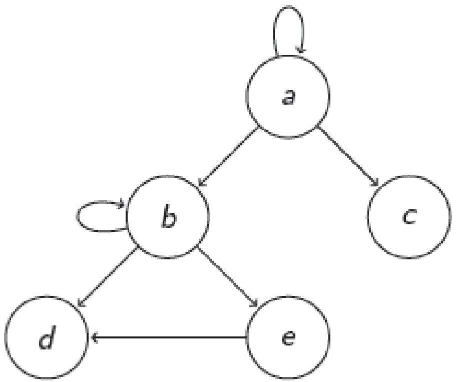
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.



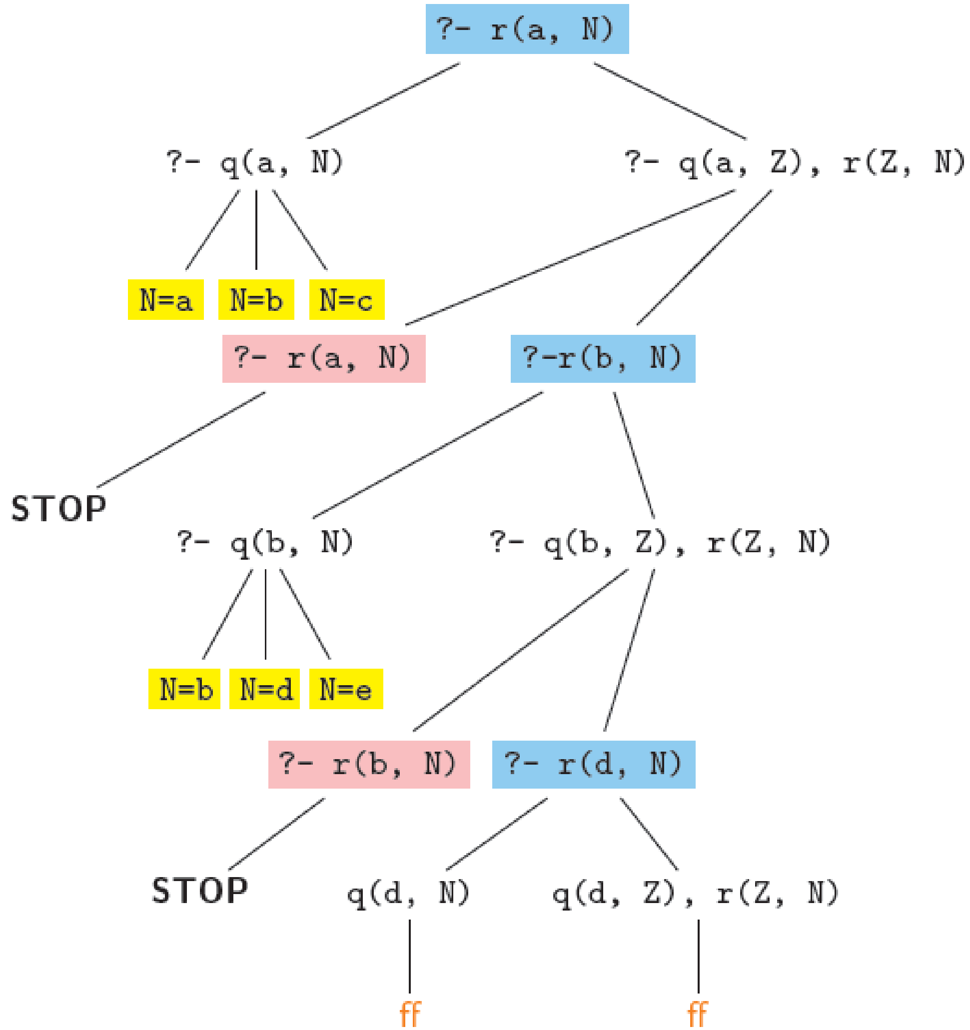

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).


r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```



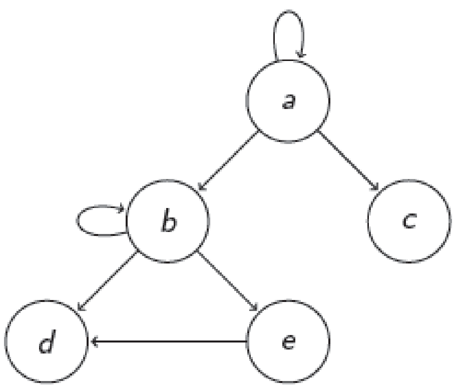
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.



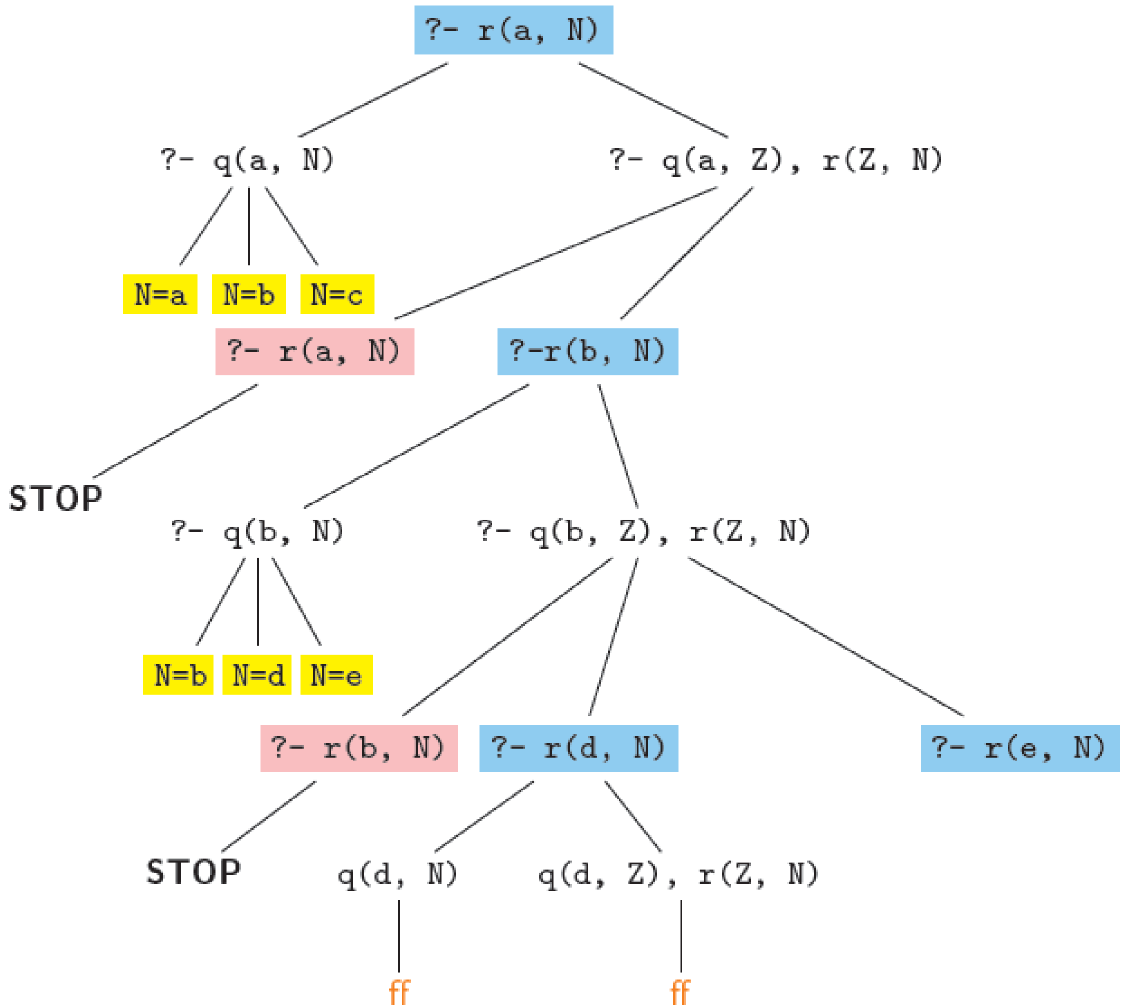

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).


r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```



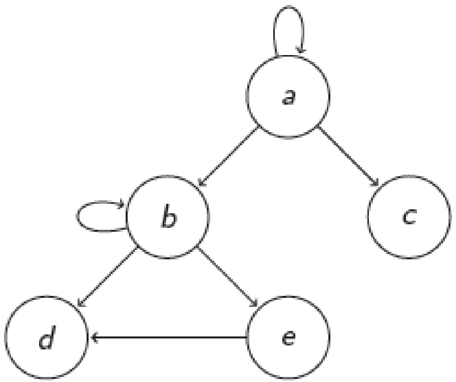
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.



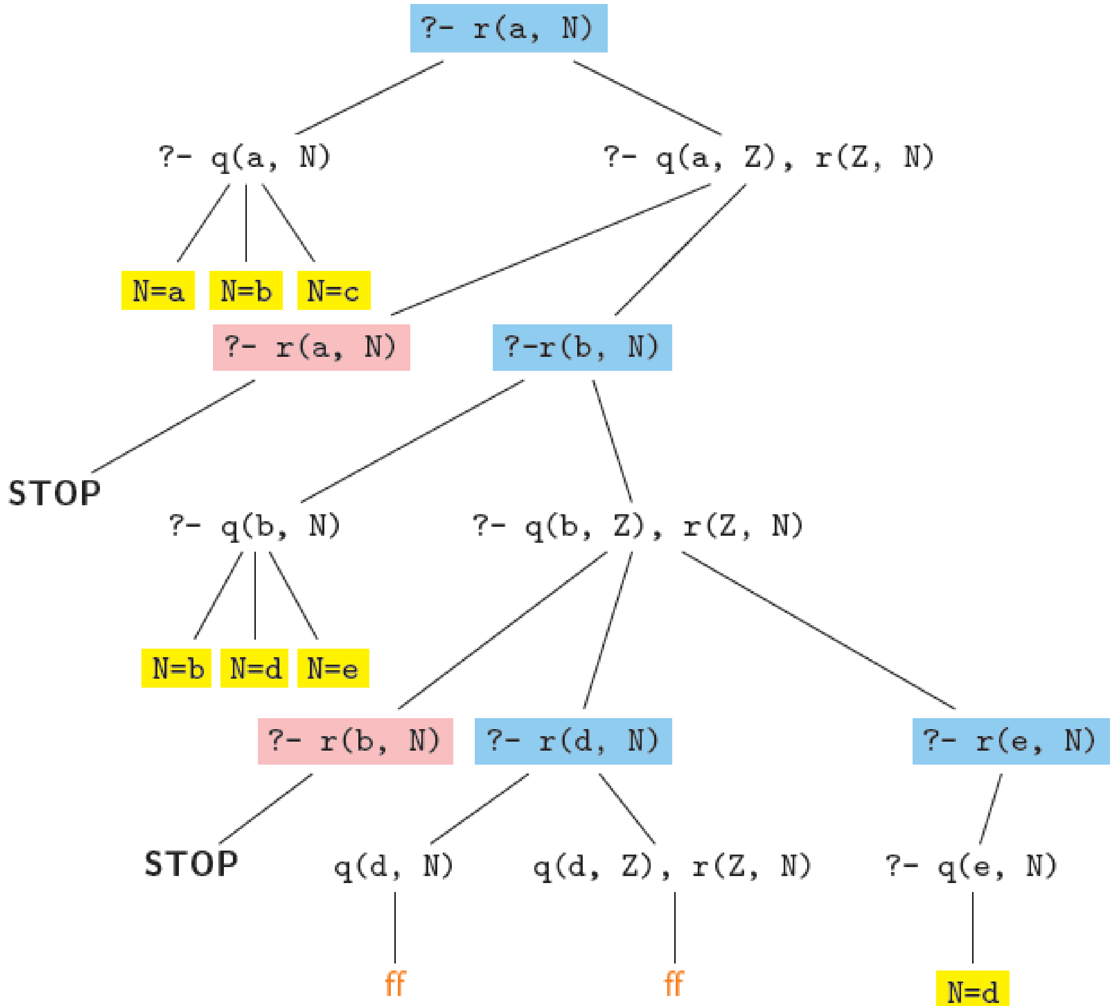

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).


r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```



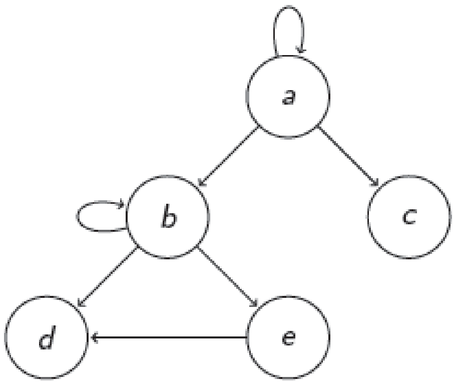
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.

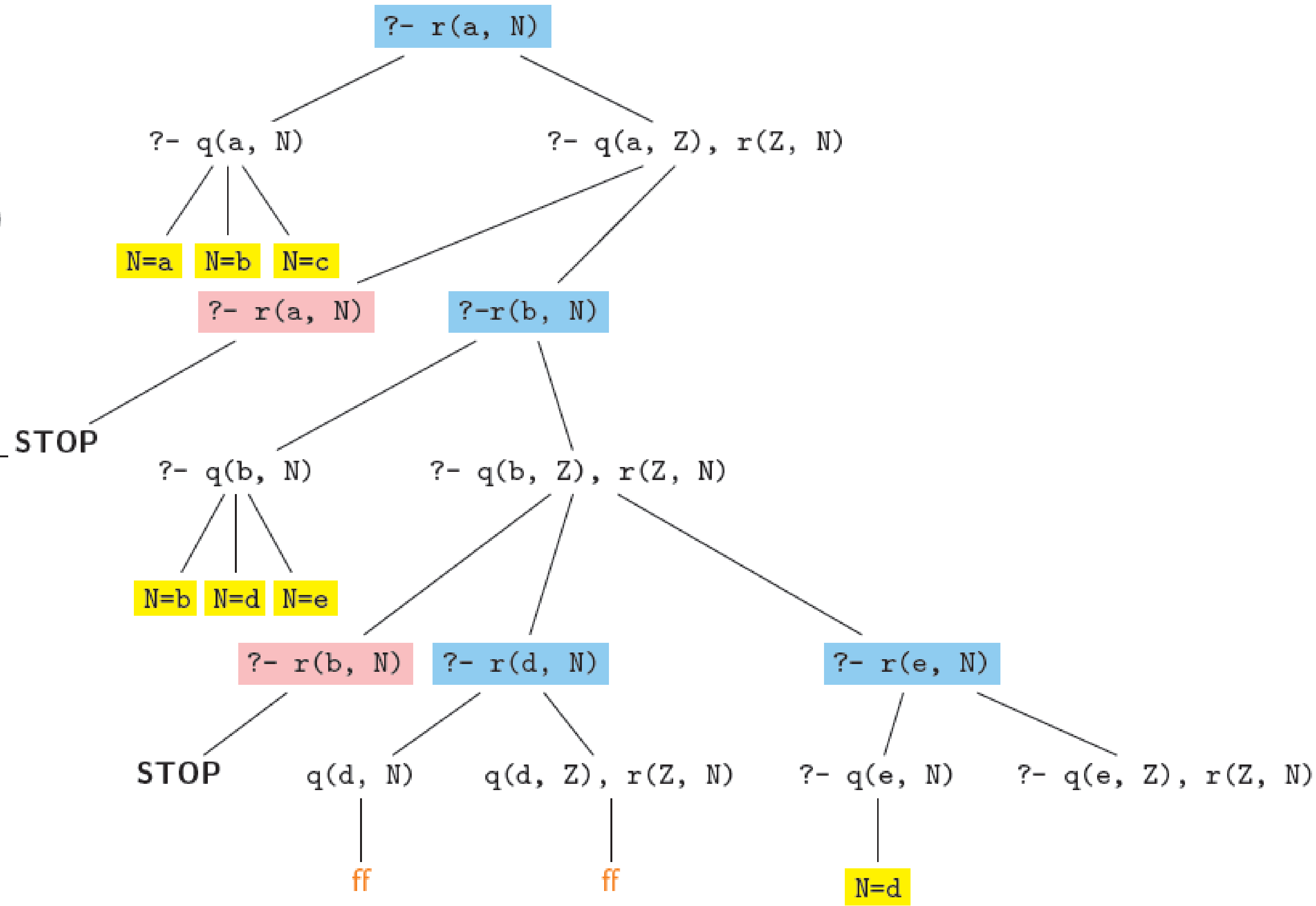



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).


r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```



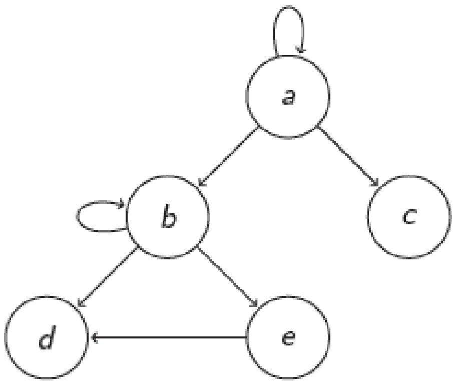
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.

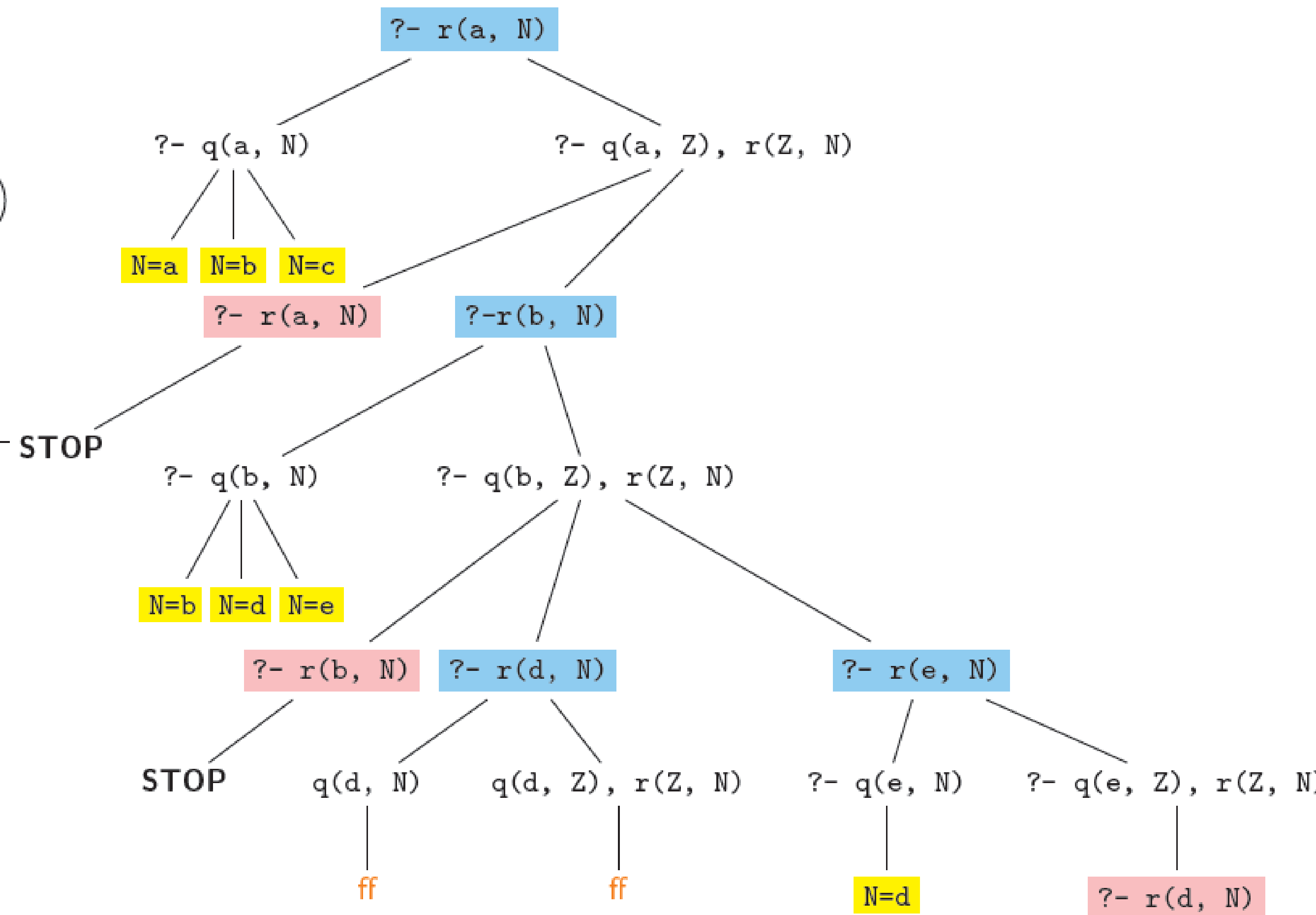





```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).


r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

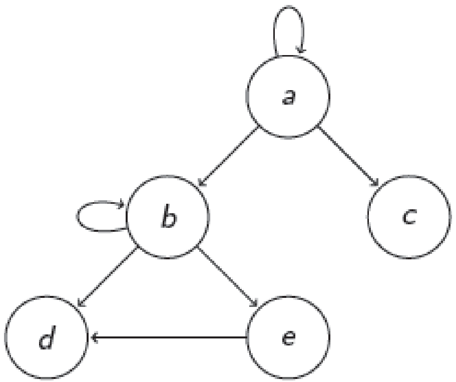
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.



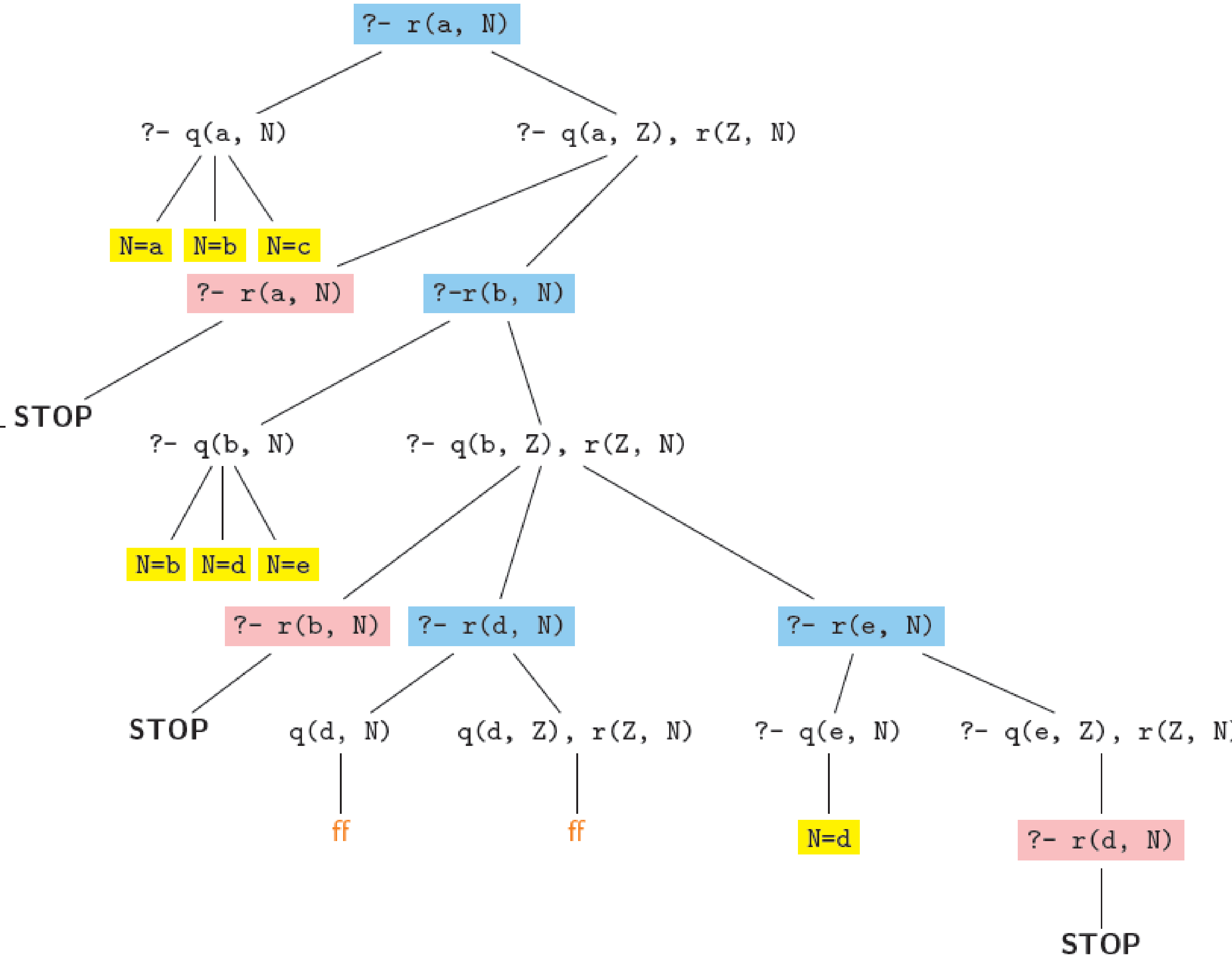



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

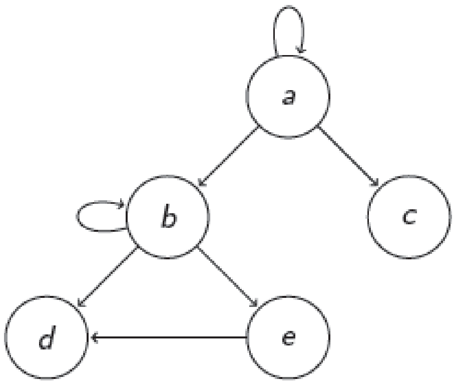
Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before.



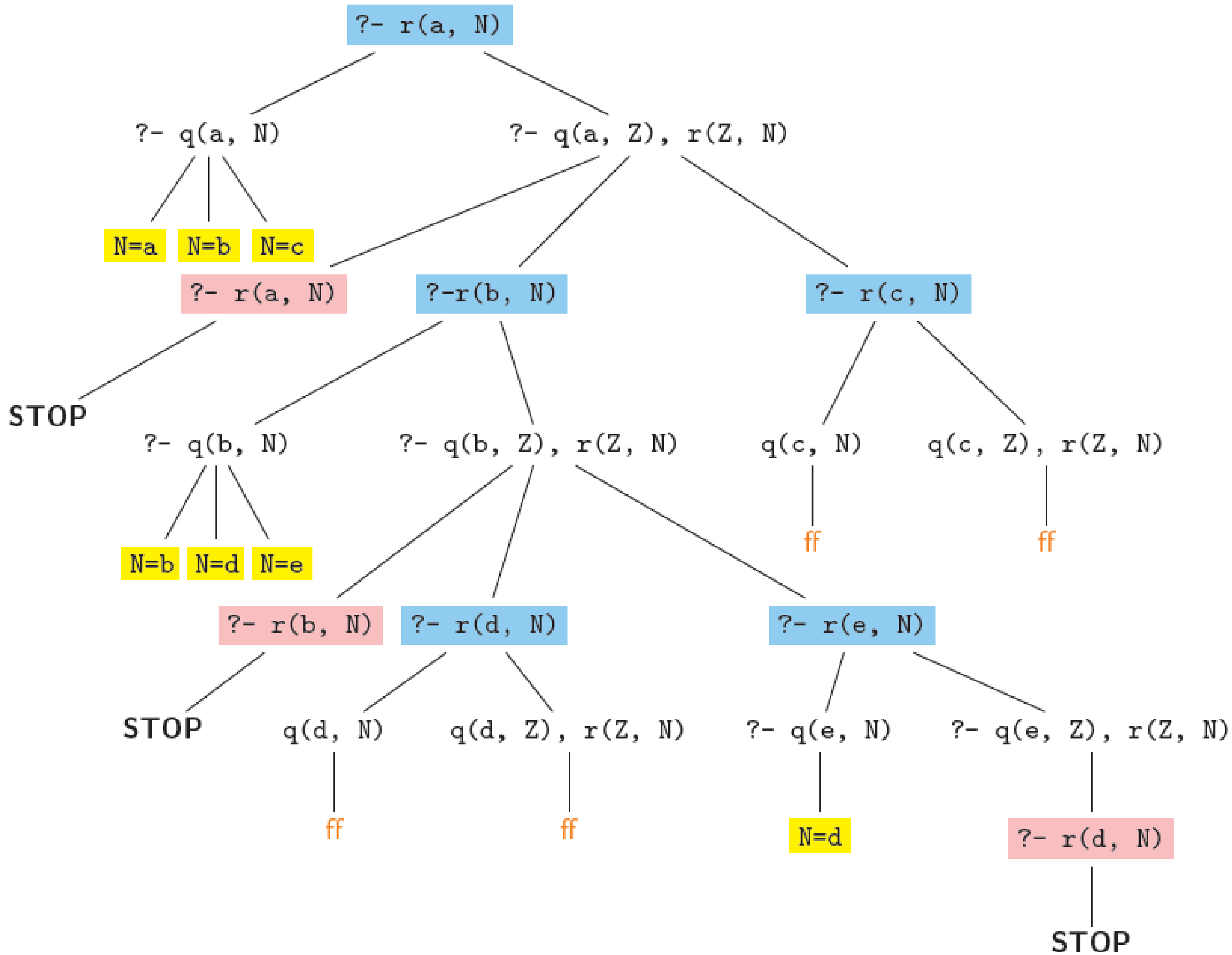

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).


r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```



Depth-First Expansion of OLD tree with a little twist: Stop if a goal has been seen before



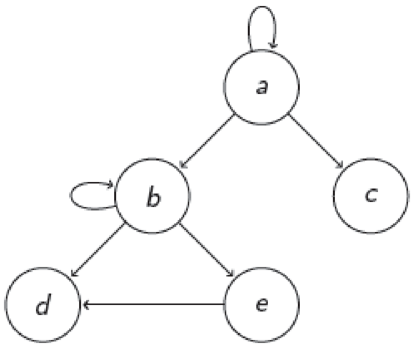

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).


r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```



Rationale for goal-based stopping

- The OLD tree is a representation of search for successful derivations
  - which are finite sequences of goals terminating in an empty goal
- If there is a successful derivation, then there is an equivalent one that does not repeat the same goal (compare to reachability via loop-free paths in a graph)
- Hence ignoring paths with repeated goals is ***sound***: the derivations pruned away by stopping have equivalent ones that will not be ignored
- Unfortunately, this scheme still does not fix the problem of infinite derivations

Infinite Derivations Despite Stopping Condition

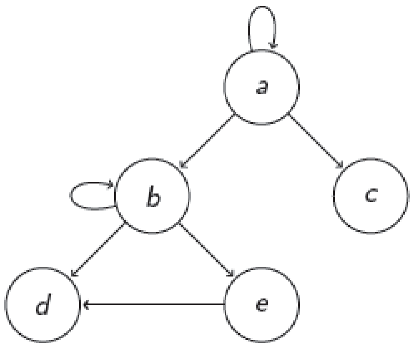


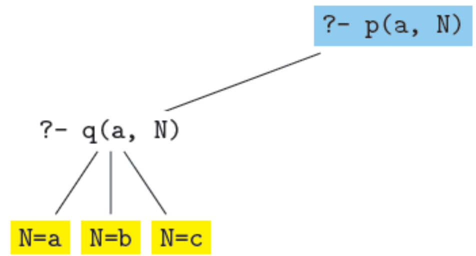

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

```
%Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).
```

```
?- p(a, N)
```

Infinite Derivations Despite Stopping Condition



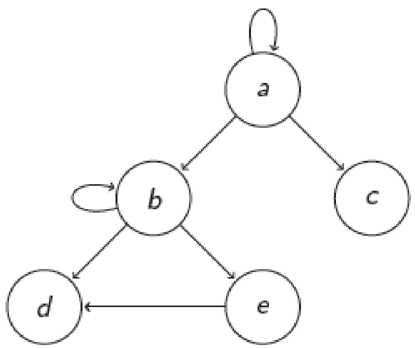



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

Expand tree as usual

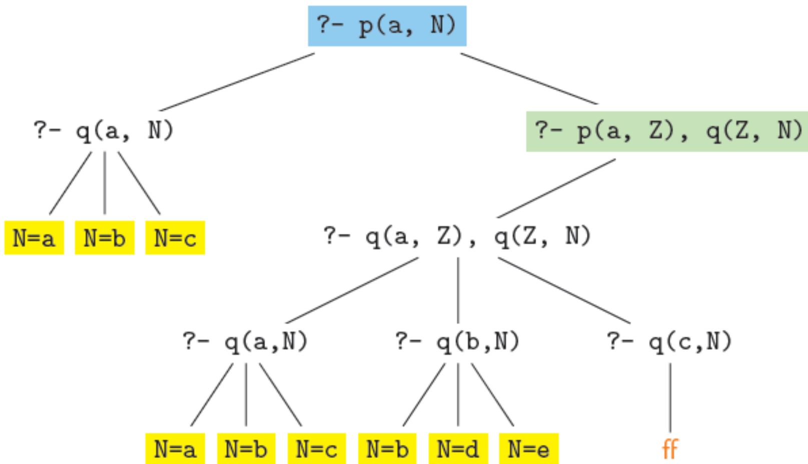
```
%Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).
```

Infinite Derivations Despite Stopping Condition



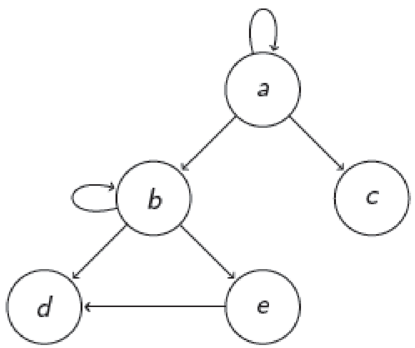

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

```
%Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).
```

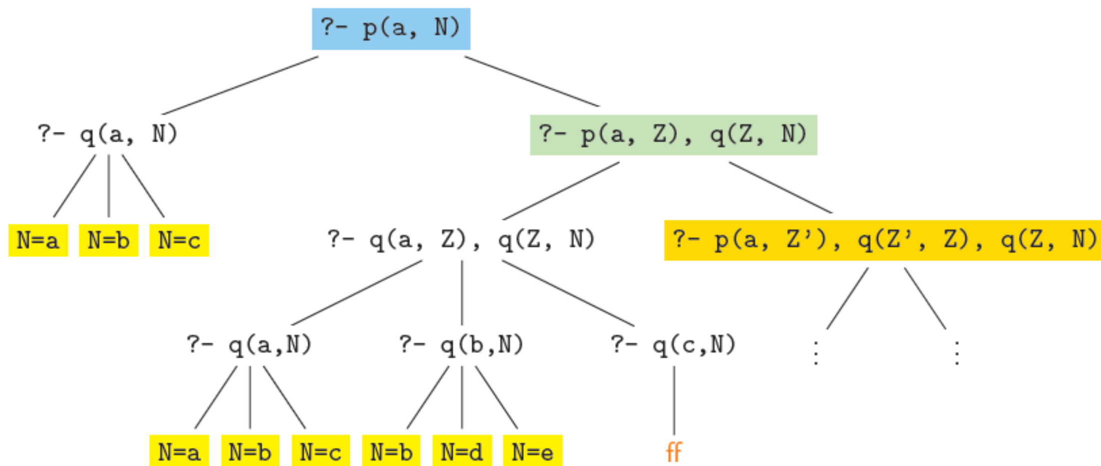


Expand tree as usual

Infinite Derivations Despite Stopping Condition




```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

%Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).
```

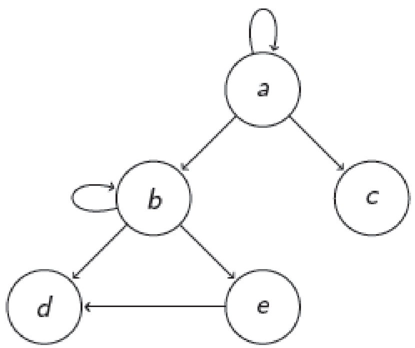


Note that the right-most branch has ever-growing goals!

OLD Resolution with Tabling (OLDT)

- The selected **literal** at a step in a derivation is known as a ***call*** **(only individual literals are tabled)**
- OLDT maintains a **table of calls** (initially empty)
- With each call, it maintains a table of computed answers (initially empty)
- Start resolution as in OLD
- When a literal is selected, check the call table.
  - If the literal is in the table, **resolve it with its answers in its answer table**
  - If the literal is not in the table, resolve with program clauses (as in OLD), and **add computed answers to its answer table**

OLDT Example



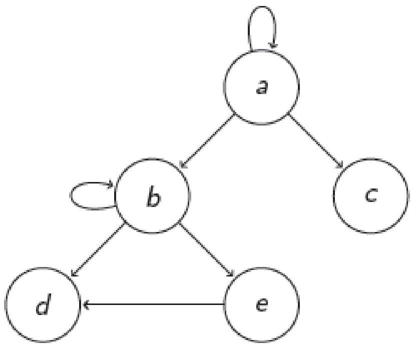

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
:- table p/2.
%Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).
```

```
?- p(a, N)
```

Calls	Answers

Start with empty tables

OLDT Example



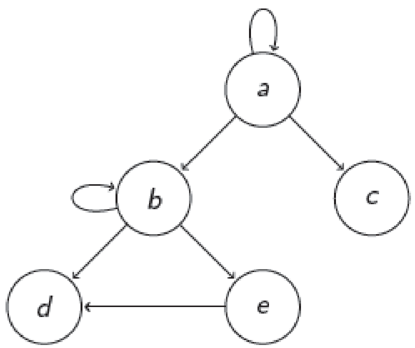

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
:- table p/2.
%Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).
```

?- $\boxed{\texttt{p(a, N)}}$

Calls	Answers

Pick selected literal. Is it in call table?

OLDT Example



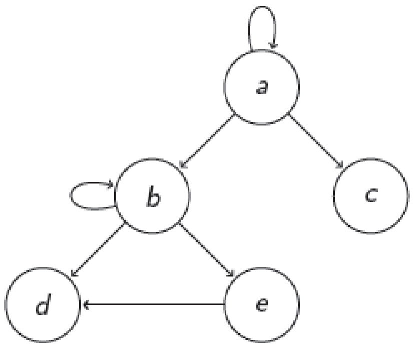

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
:- table p/2.
%Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).
```

```
?- p(a, N)
```

Calls	Answers
p(a, W)	

Add to call table

OLDT Example




```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

:- table p/2.

%Alternative formulation
% of reachability: Note
% use of LEFT recursion

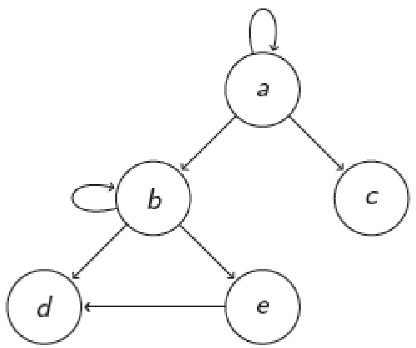
```
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).
```

```
?- p(a, N)
?- q(a, N)
```

Calls	Answers
p(a, W)	

Do OLD resolution with program clauses

OLDT Example



?- p(a, N)

?- q(a, N)

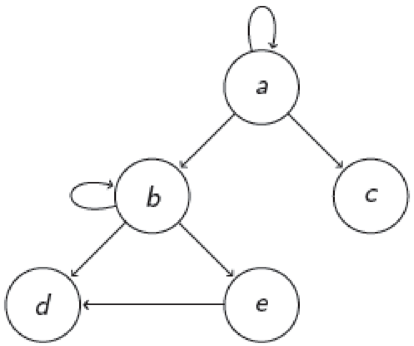
N=a

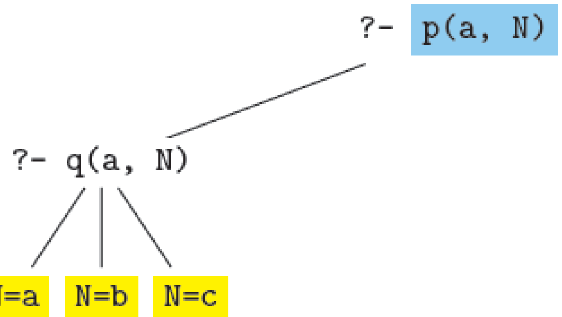
```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
:- table p/2.
%Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).
```

Calls	Answers
p(a, W)	{p(a,a)

Add computed answer to table (if not already there)

OLDT Example



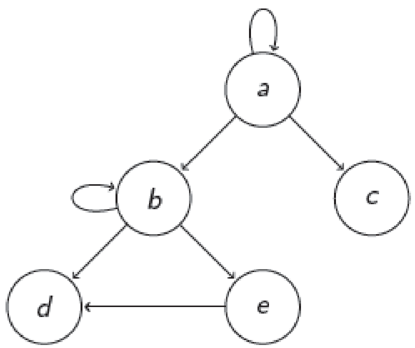



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
:- table p/2.
%Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).
```

Calls	Answers
p(a, W)	{p(a,a), p(a,b), p(a,c)

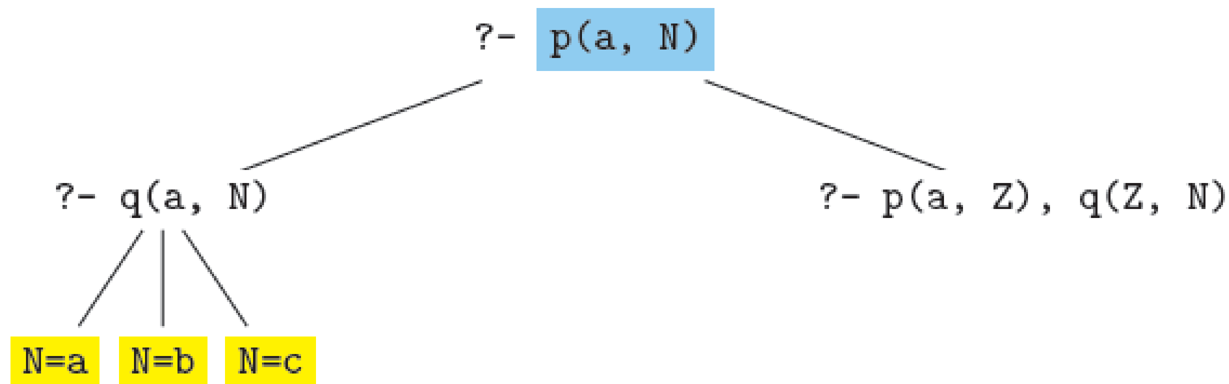
Add computed answer to table (if not already there)

OLDT Example




```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

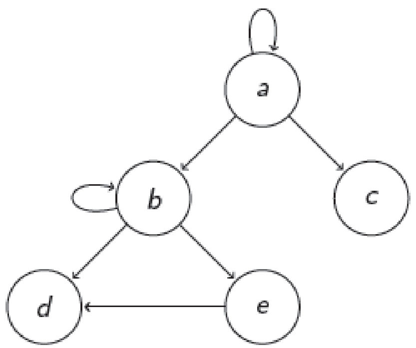
```
:- table p/2.
%Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).
```

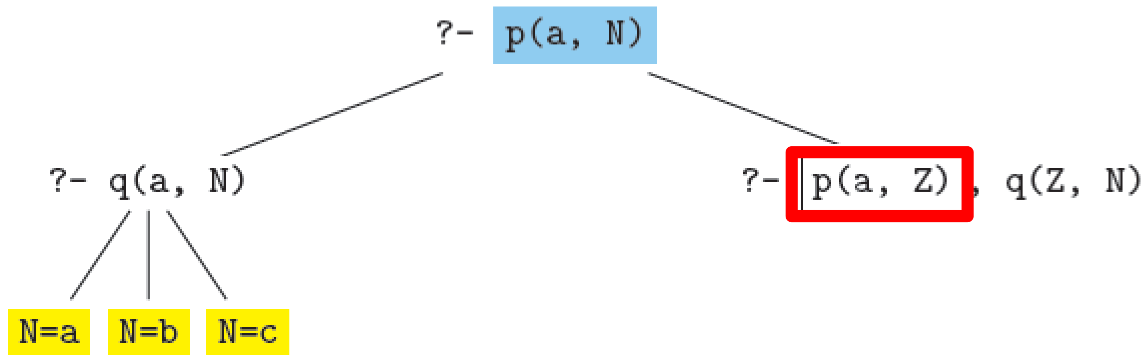


Calls	Answers
p(a, W)	{p(a,a), p(a,b), p(a,c)

Continue with OLD resolution

OLDT Example



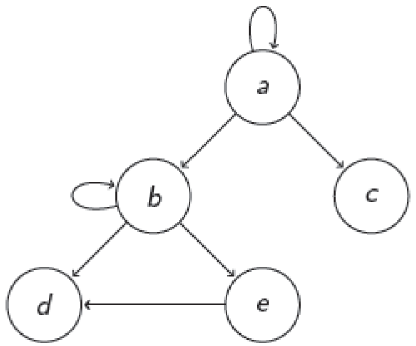


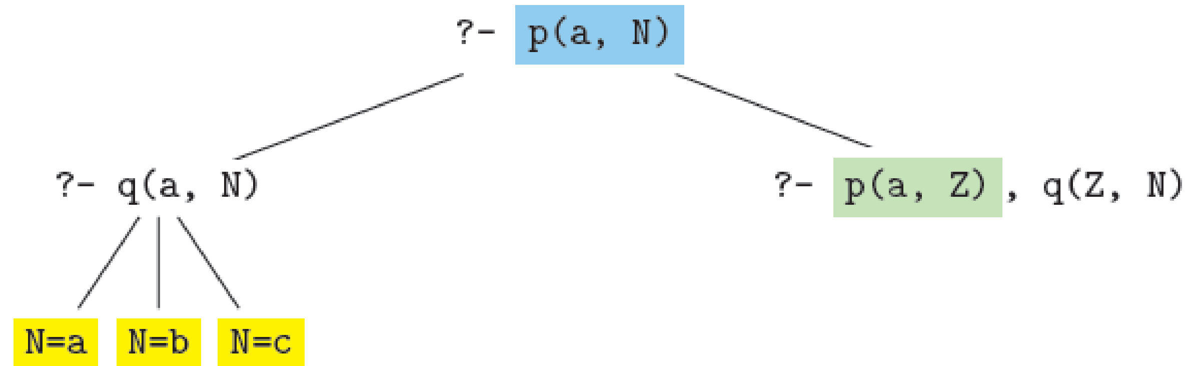

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
:- table p/2.
%Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).
```

Calls	Answers
p(a, W)	{p(a,a), p(a,b), p(a,c)

Pick selected literal. Is it in call table?

OLDT Example



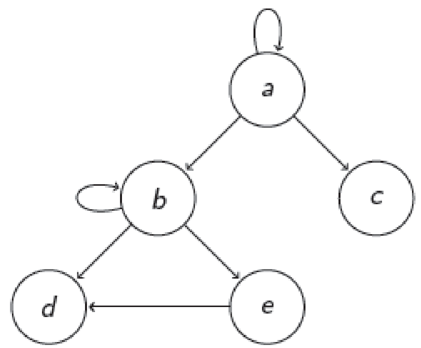


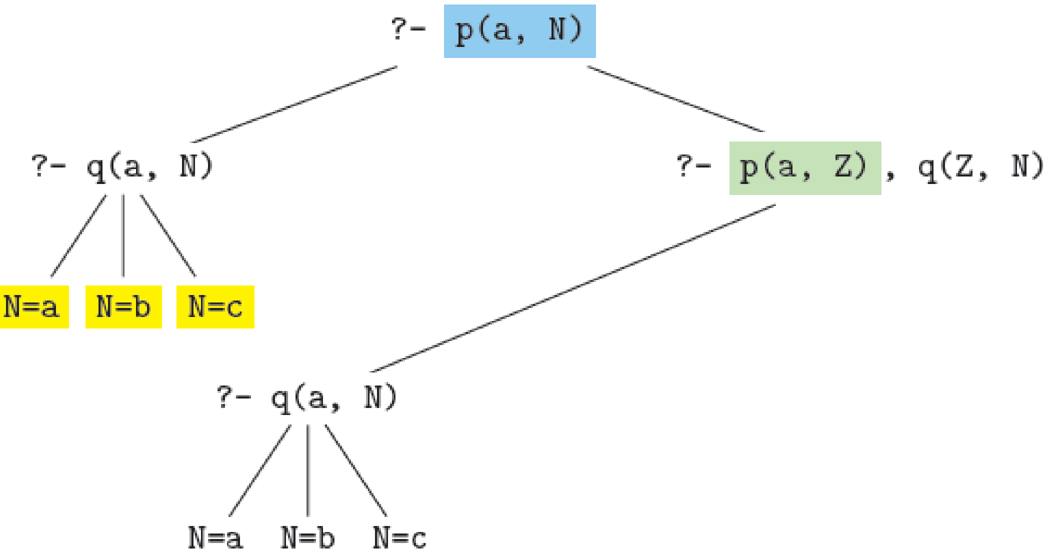

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
:- table p/2.
%Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).
```

Calls	Answers
p(a, W)	{p(a,a), p(a,b), p(a,c)

Yes, resolve with answers in table

OLDT Example



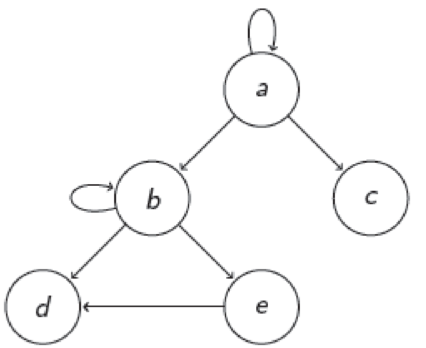



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
:- table p/2.
%Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).
```

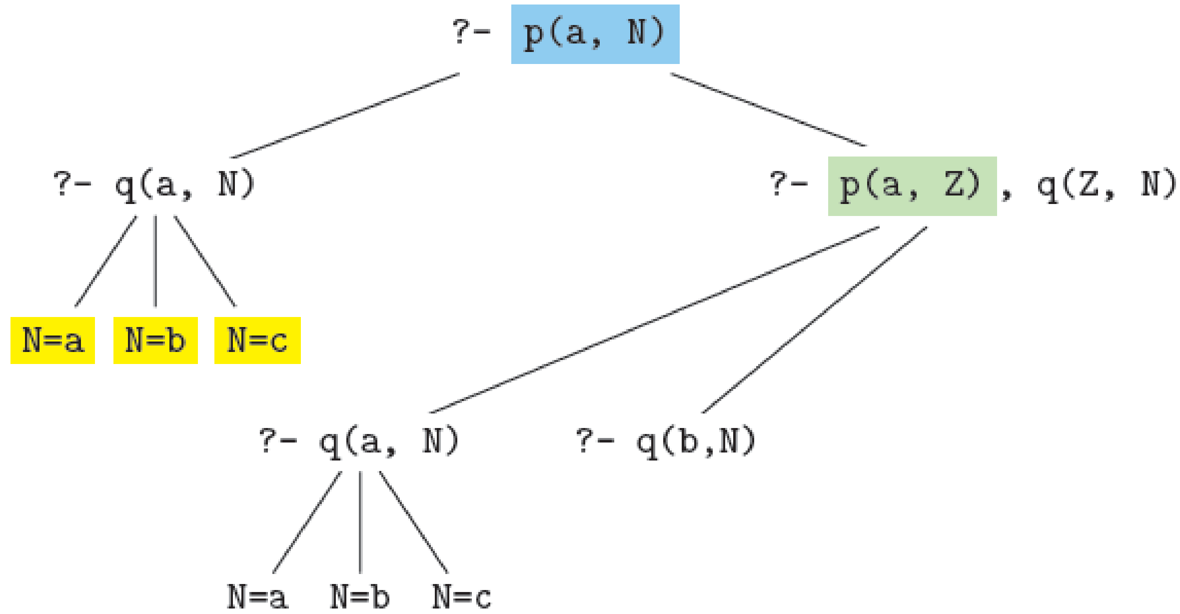
Calls	Answers
p(a, W)	{p(a,a), p(a,b), p(a,c)

Add computed answer to table (if not already there)

OLDT Example



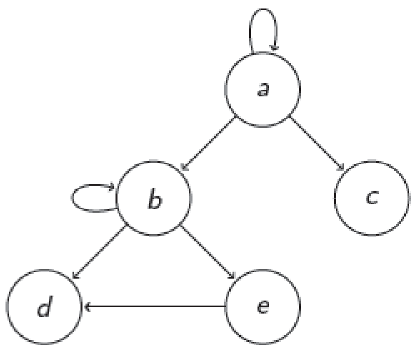

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
:- table p/2.
%Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).
```



Calls	Answers
p(a, W)	{p(a,a), p(a,b), p(a,c)

Continue resolving with answers in table

OLDT Example

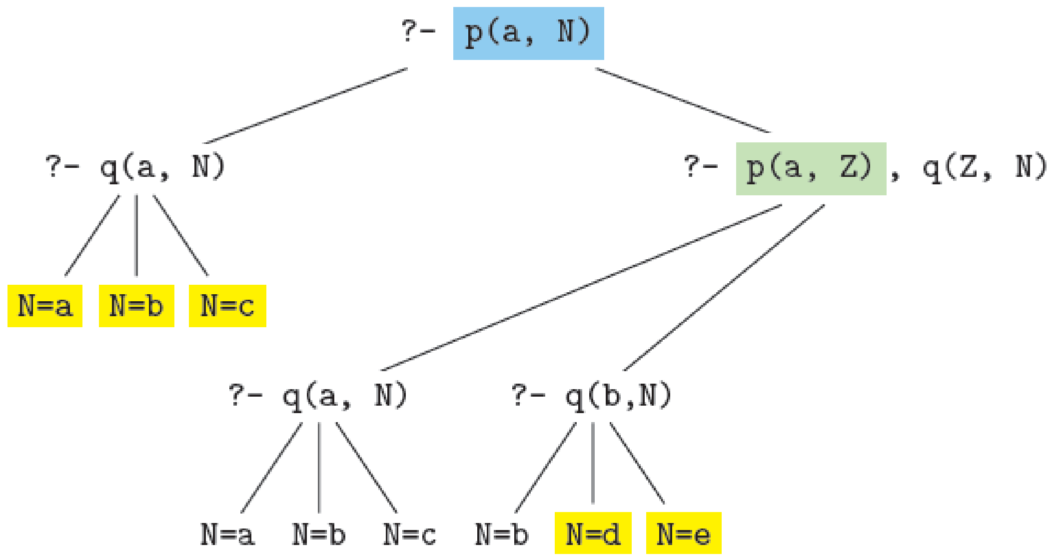



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

```
:- table p/2.
```

%Alternative formulation
% of reachability: Note
% use of LEFT recursion

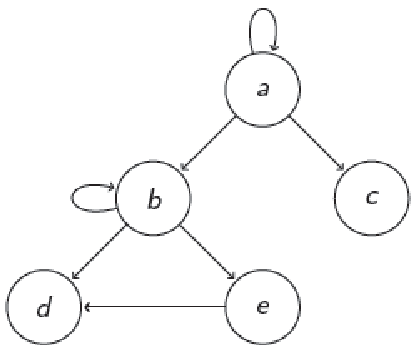
```
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).
```



Calls	Answers
p(a, W)	{p(a,a), p(a,b), p(a,c) p(a,d), p(a,e)

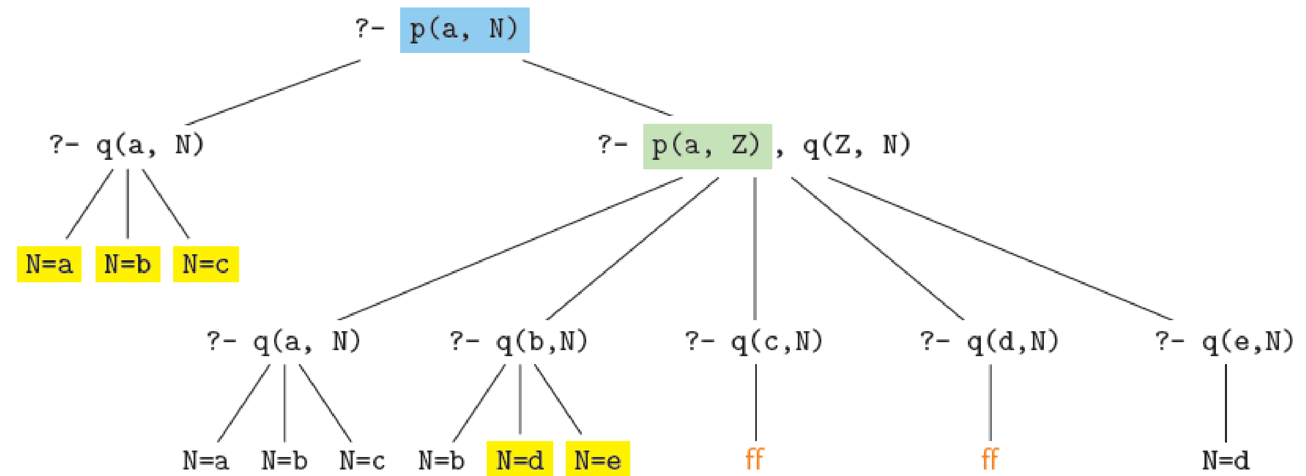
Add computed answer to table (if not already there)

OLDT Example




```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

```
:- table p/2.
%Alternative formulation
% of reachability: Note
% use of LEFT recursion
p(X,Y) :- q(X,Y).
p(X,Y) :-
    p(X,Z), q(Z,Y).
```



Calls	Answers
p(a, W)	{p(a,a), p(a,b), p(a,c) p(a,d), p(a,e)}

Complete when all answers have been considered for resolution

OLDT Forest

- When we get new answer, we will have to return to previous queries to continue their execution
- When a literal is selected, mark it as a ***consumer***
- Check the call table:
  - If the literal is not in the table, **start a new tree for that literal** with its root marked as ***generator***
  - **Resolve generator with program clauses** (as in OLD), and add computed answers to its answer table
- Resolve consumer with answers in its generator's table

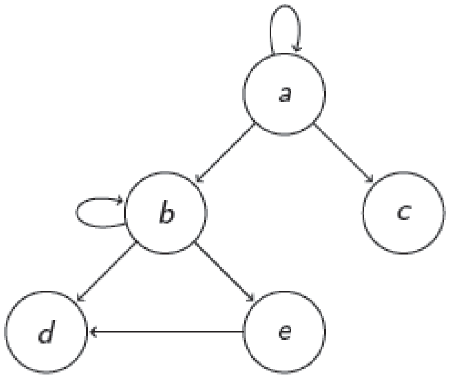
Calls and answers in tables

- Calls in table are standardized apart: i.e. their variables are renamed so that they are not identical to any other variable
- Answers in a call's computed answer table share variables with their call
- When checking if a literal **`l`** is in call table:
  - We can check for ***variance***: for a call **`c`** that is identical to the given literal **`l`**, modulo names of variables
    - All answers to **`c`** are answers to **`l`**, and vice versa
  - We can check for ***subsumption***: for a call **`c`** that is more general than a given literal **`l`**, . i.e. if there is a substitution θ such that $c\theta = l$
    - Not all answers to **`c`** may be answers to **`l`** , but every answer to **`l`** is an answer to **`c`**

Notes on OLDT

- We can selectively mark which predicates we want to maintain tables for (e.g. "**`p`**" in the previous example)
  - In general, no need to maintain tables for predicates defined solely by facts (i.e. clauses with empty bodies)
- For a Datalog program, there can be only finitely many distinct calls and answers
  - So the size of tables is bounded
- The number of literals in each goal is limited by the largest clause in the program (or original goal)
- Hence for Datalog, the OLDT forest as well as table sizes are bounded

OLDT: Second Example



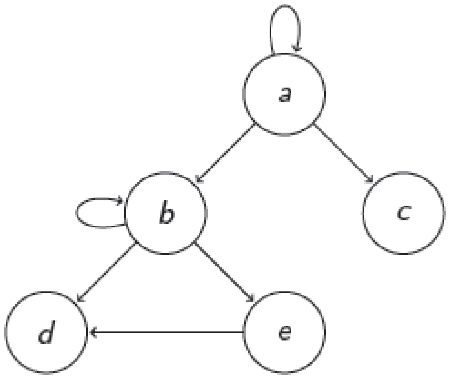
?- r(a, N)

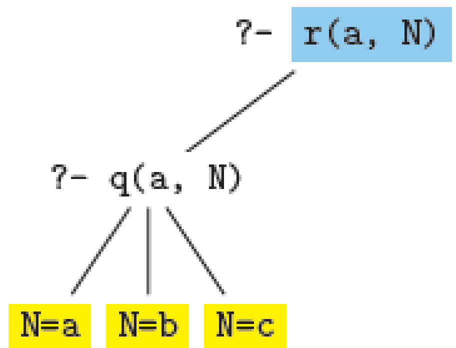
```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

```
%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

- Construct a **forest**: one tree for each call
- Root of each tree (blue) is a **generator**
- Selected literal that matches a tabled call (green) is a **consumer**

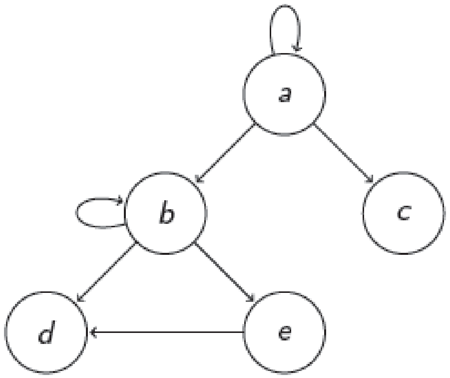
OLDT: Second Example





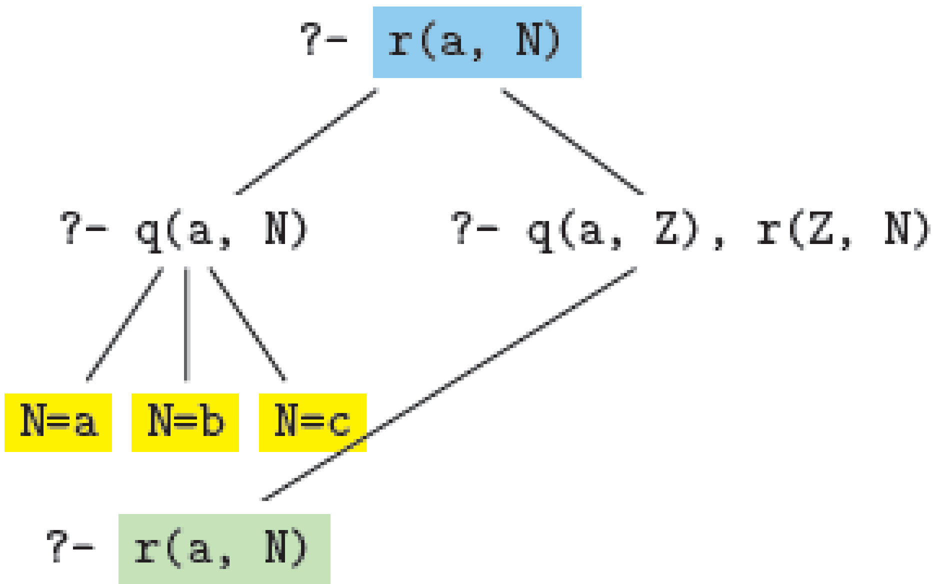

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).


%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLDT: Second Example

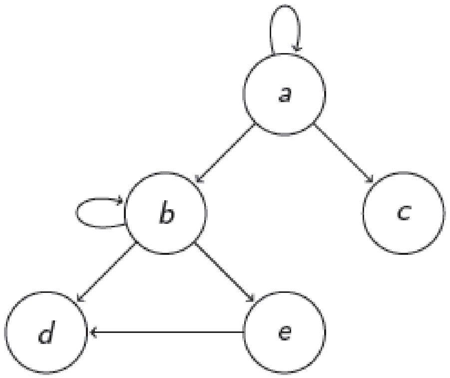


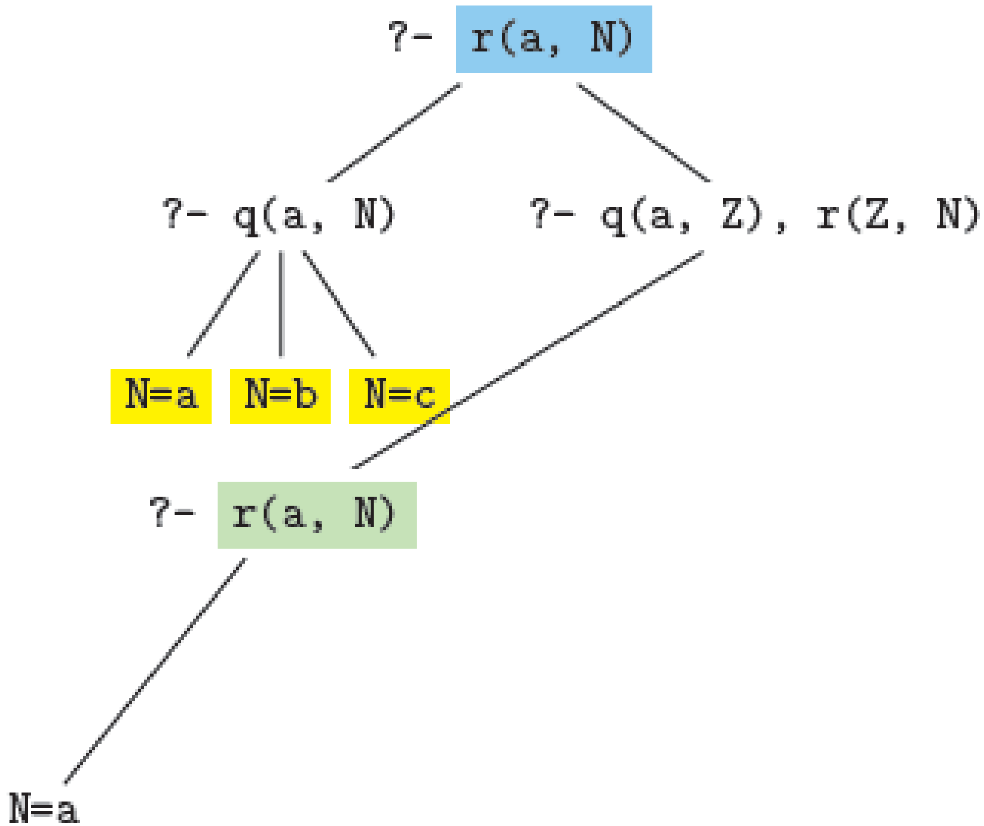

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

```
%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```



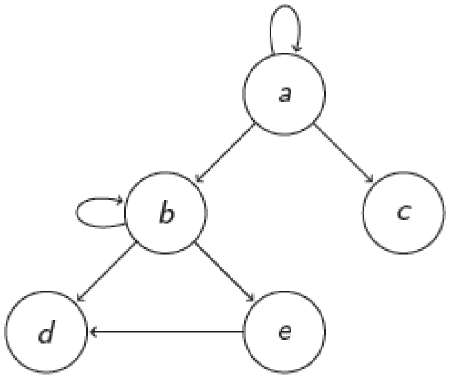
OLDT: Second Example





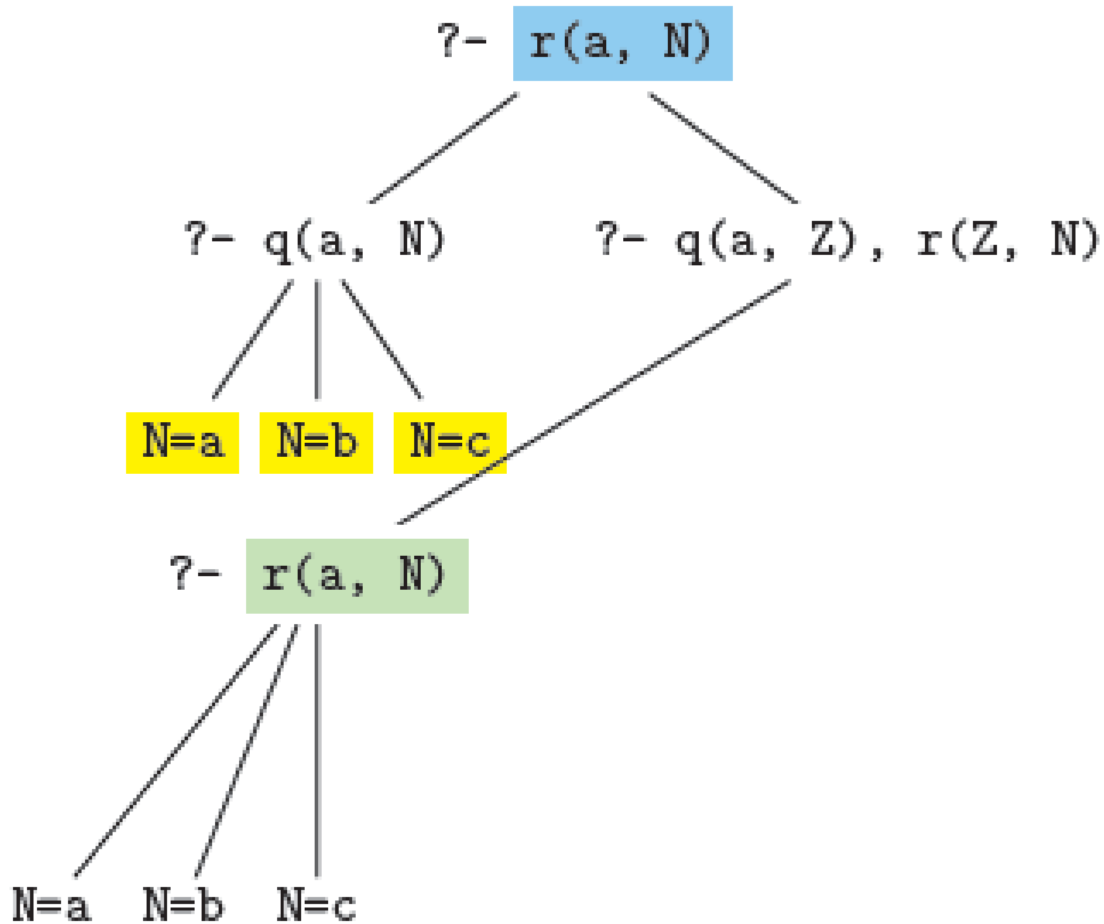

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLDT: Second Example

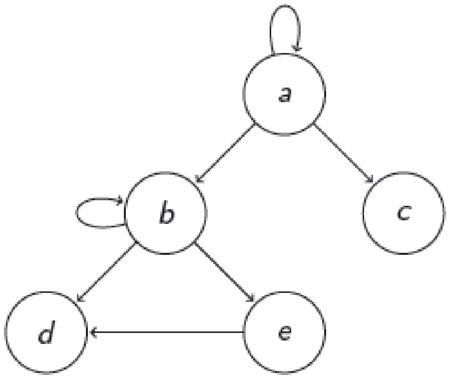



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

```
%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

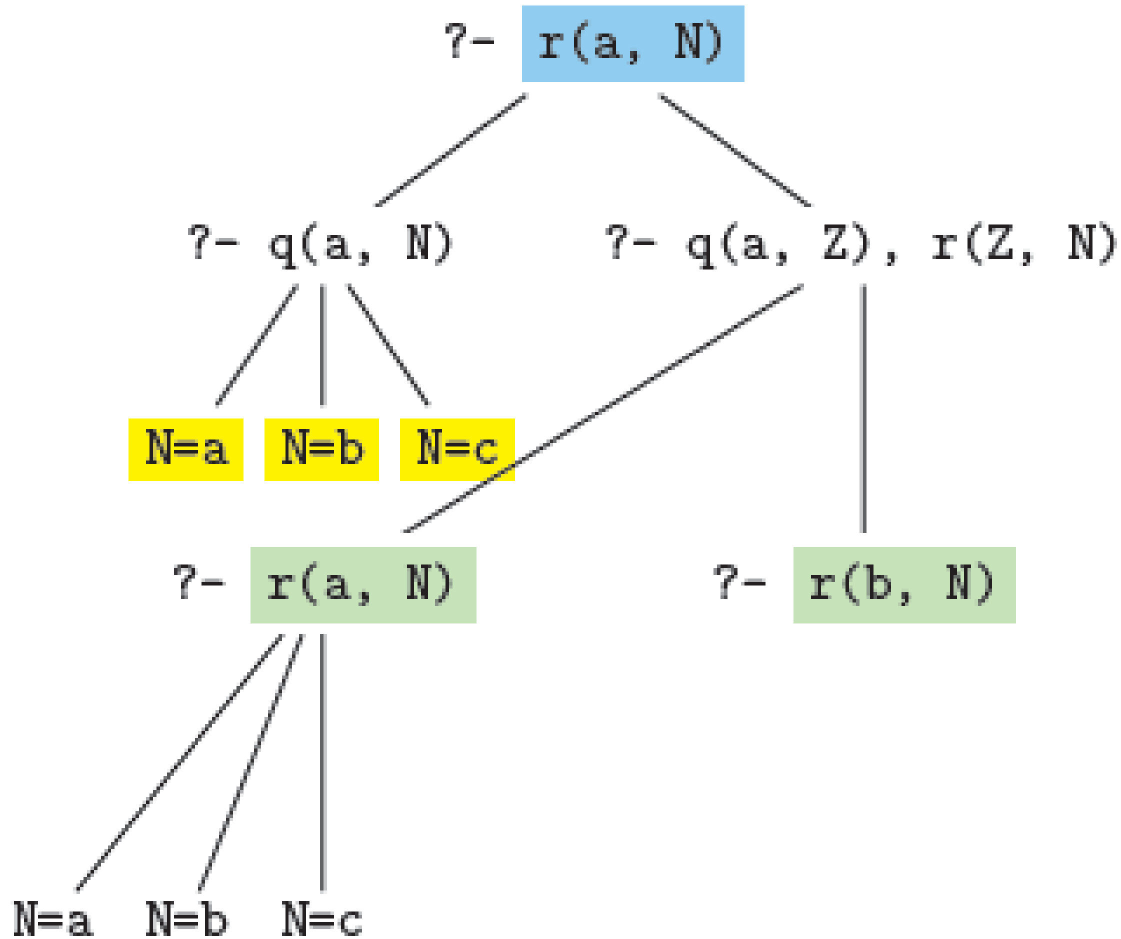


OLDT: Second Example

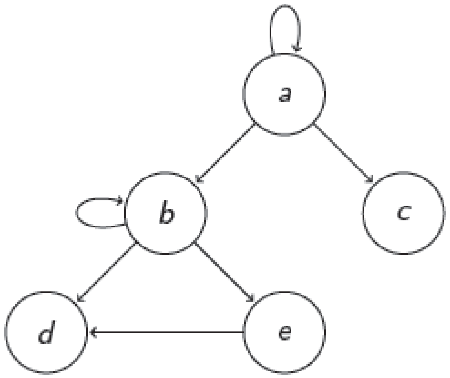



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

```
%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```



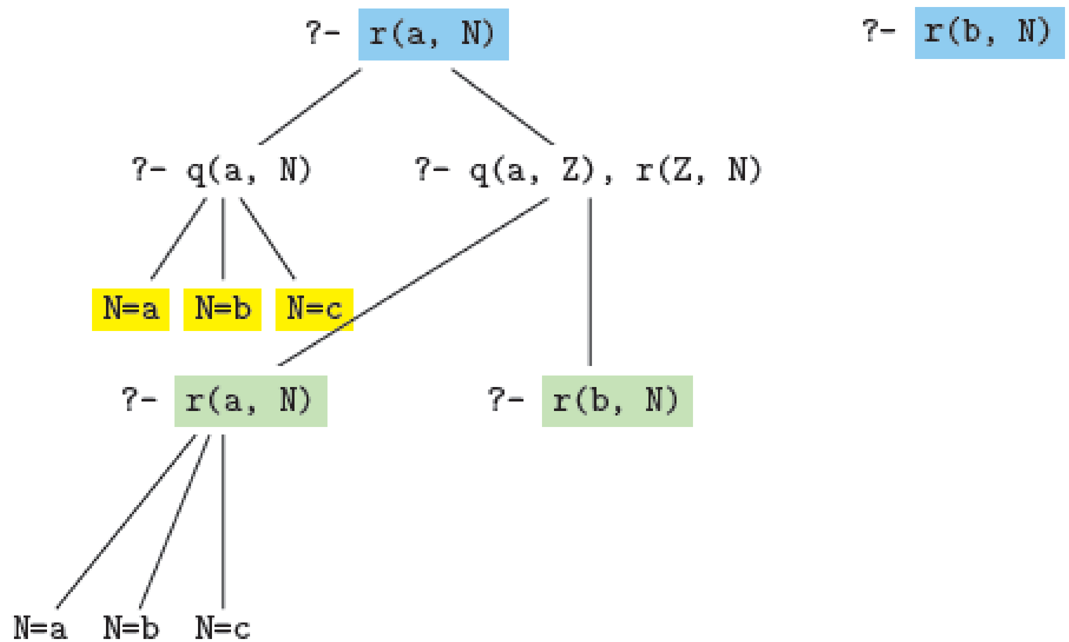
OLDT: Second Example



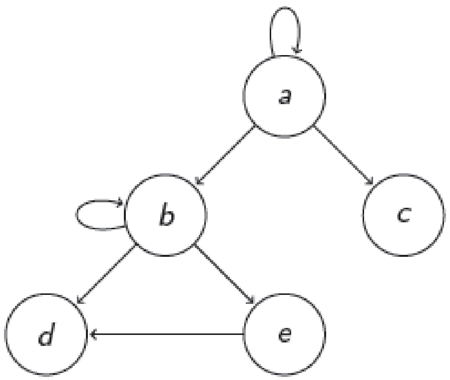

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

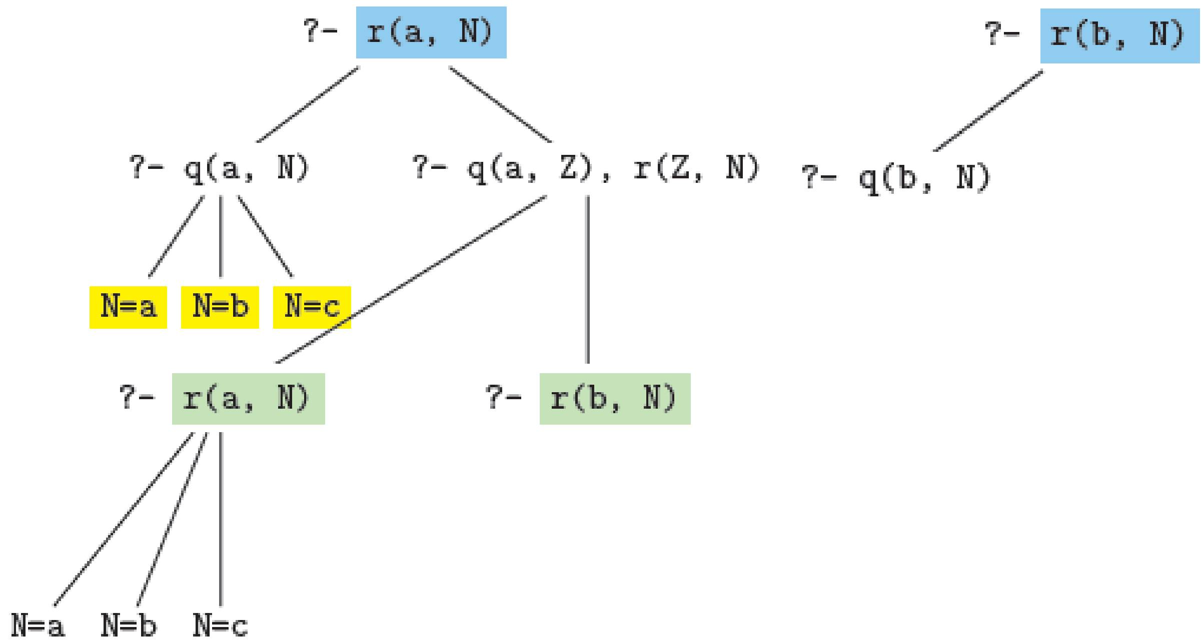
%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion

```
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```



OLDT: Second Example

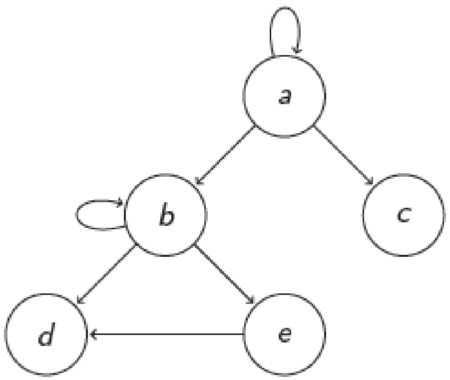


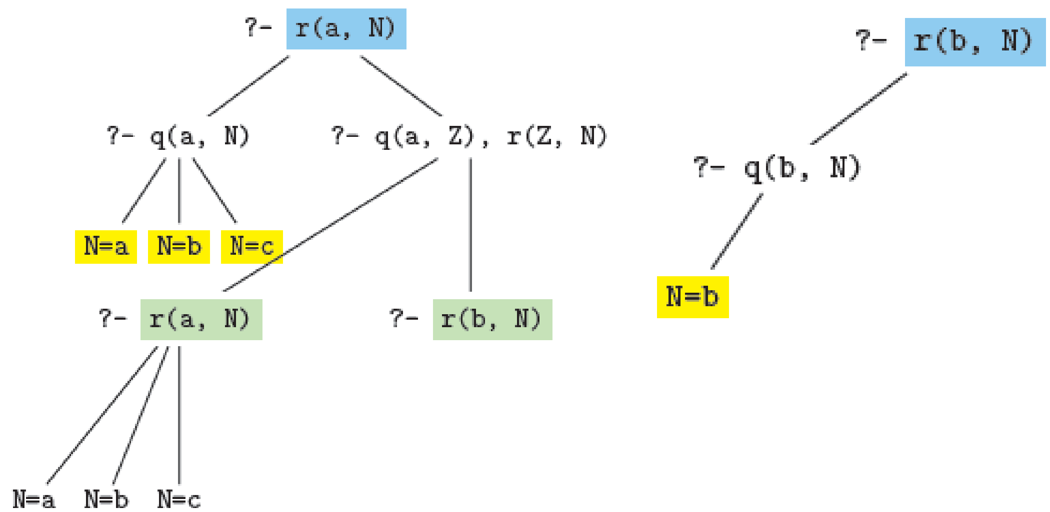



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

```
%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLDT: Second Example

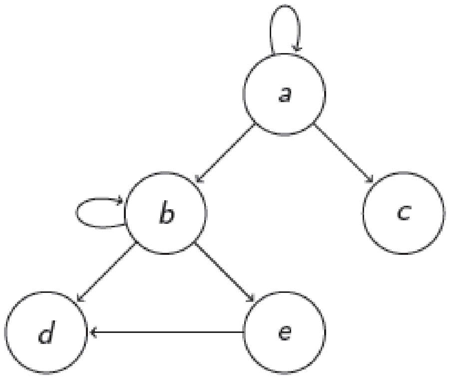


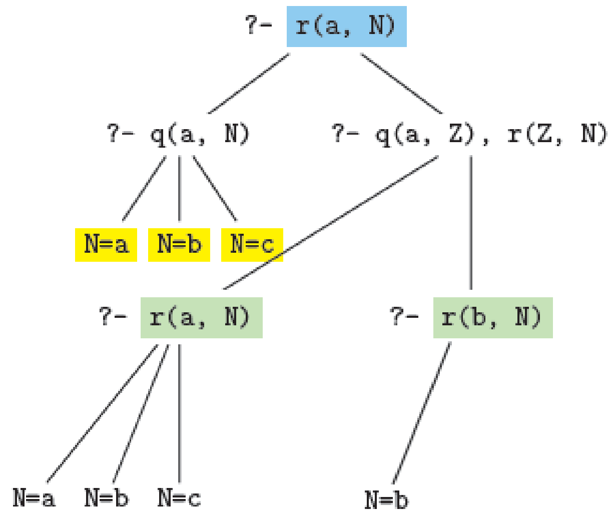


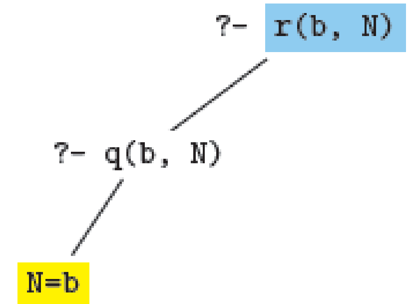

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

```
%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLDT: Second Example





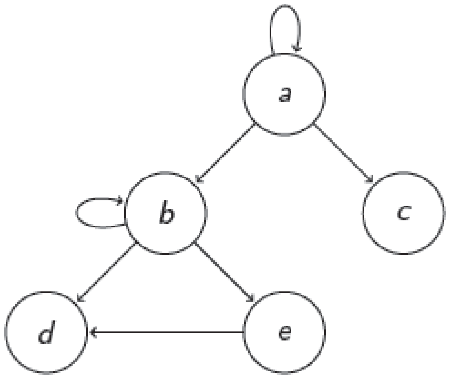



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion

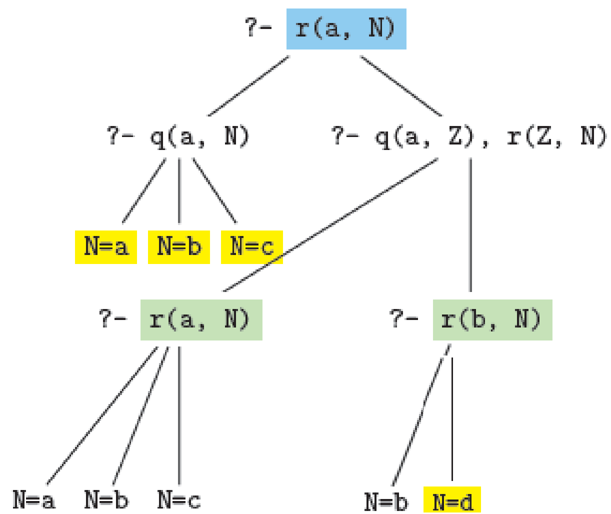
```
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

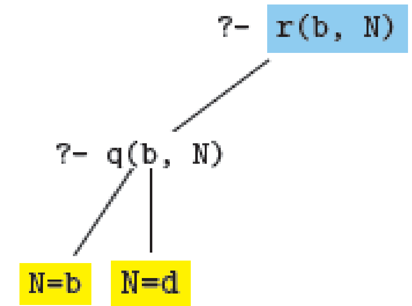
OLDT: Second Example



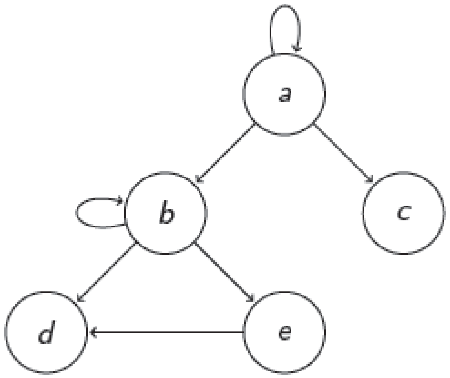

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

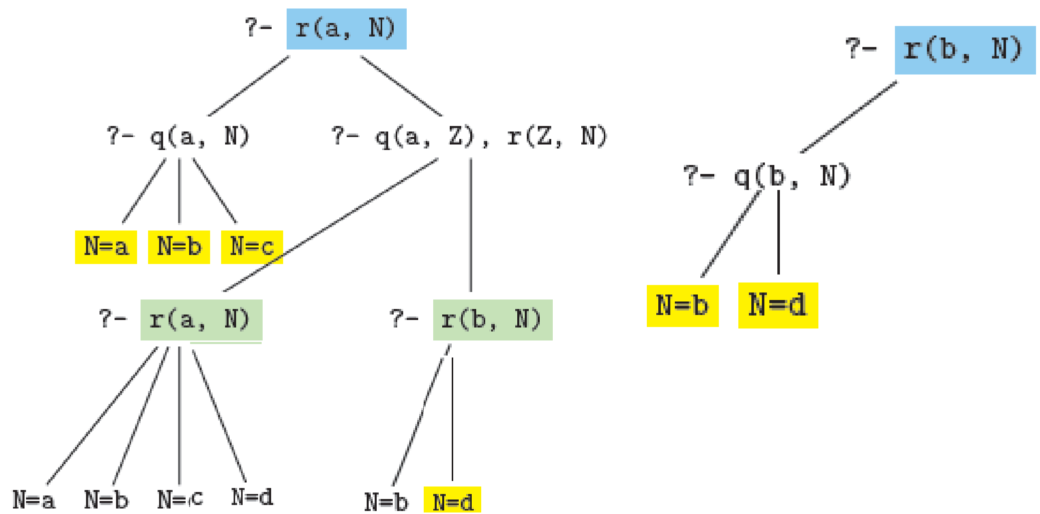
```
%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```





OLDT: Second Example

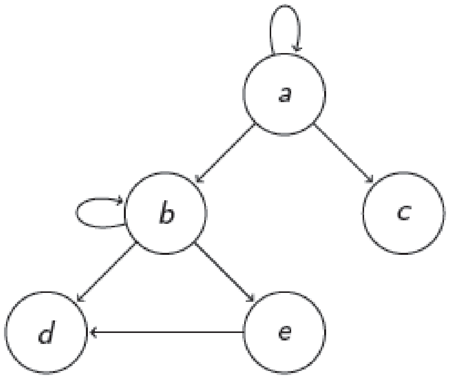


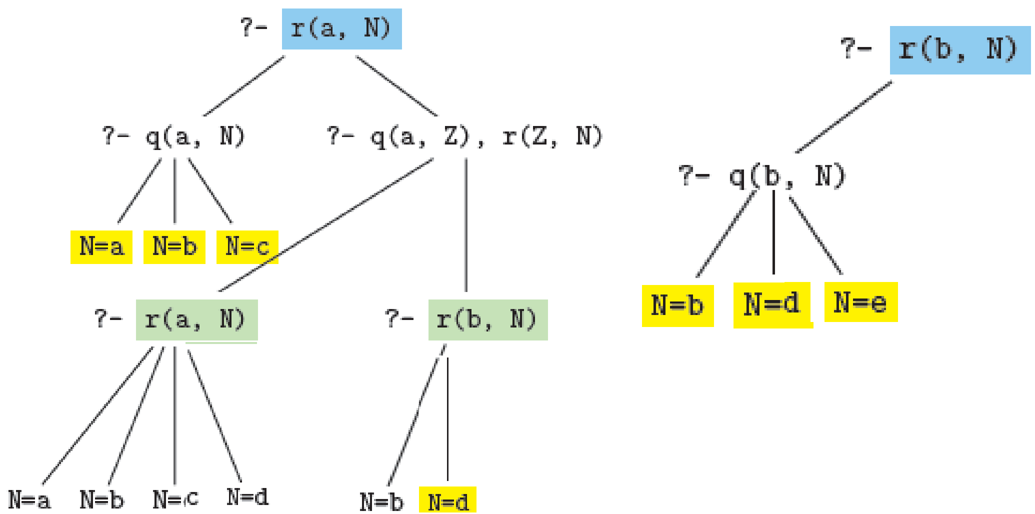



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

```
%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLDT: Second Example

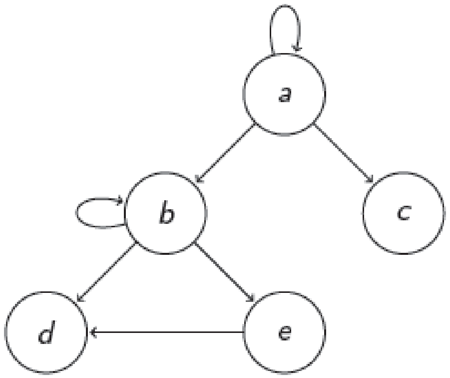


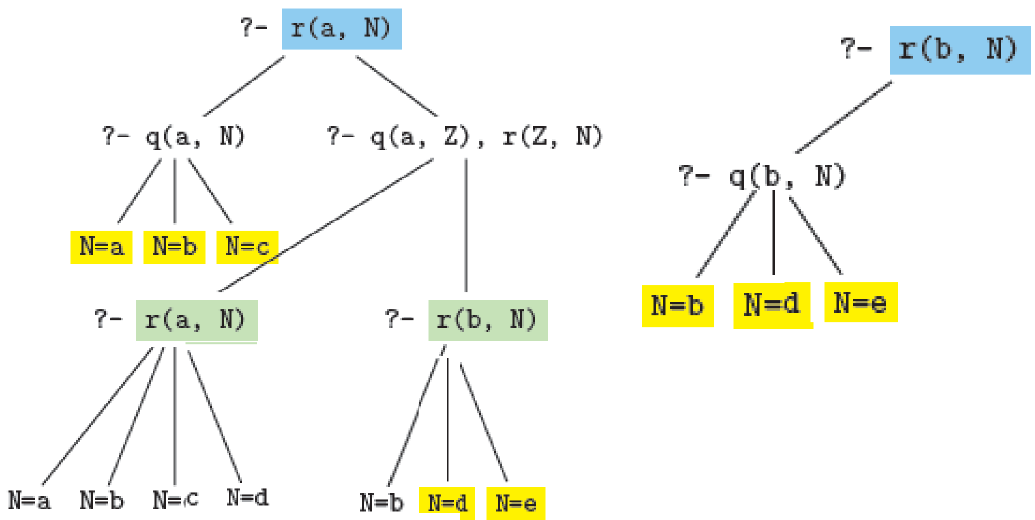



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

```
%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLDT: Second Example



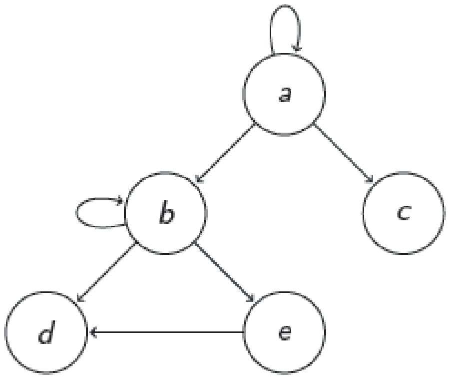


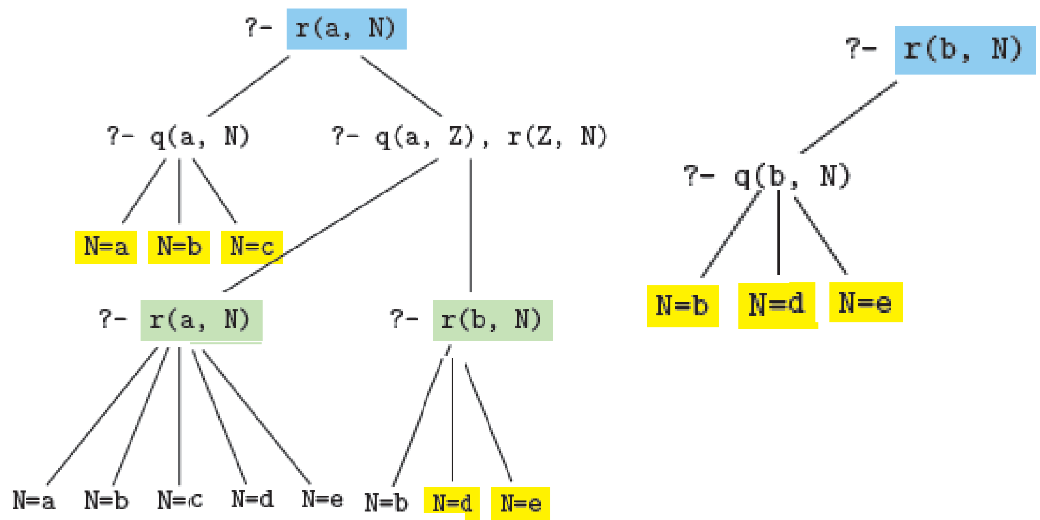

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion

```
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLDT: Second Example

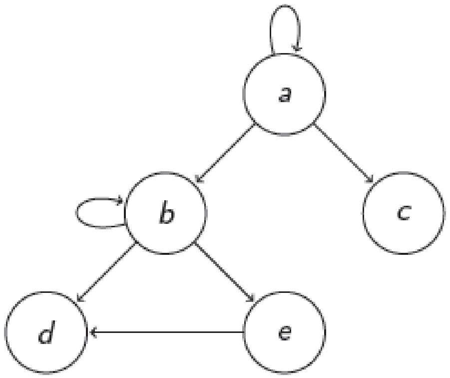


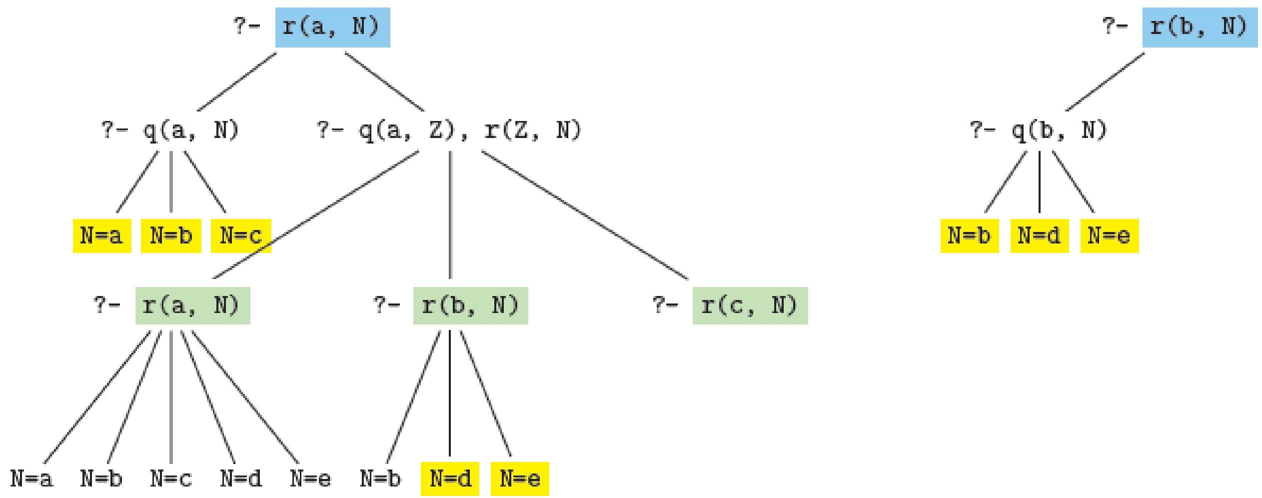



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

```
%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

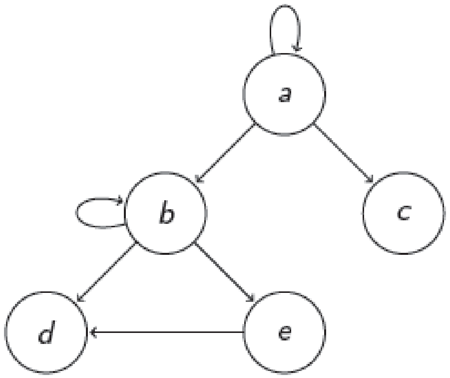
OLDT: Second Example

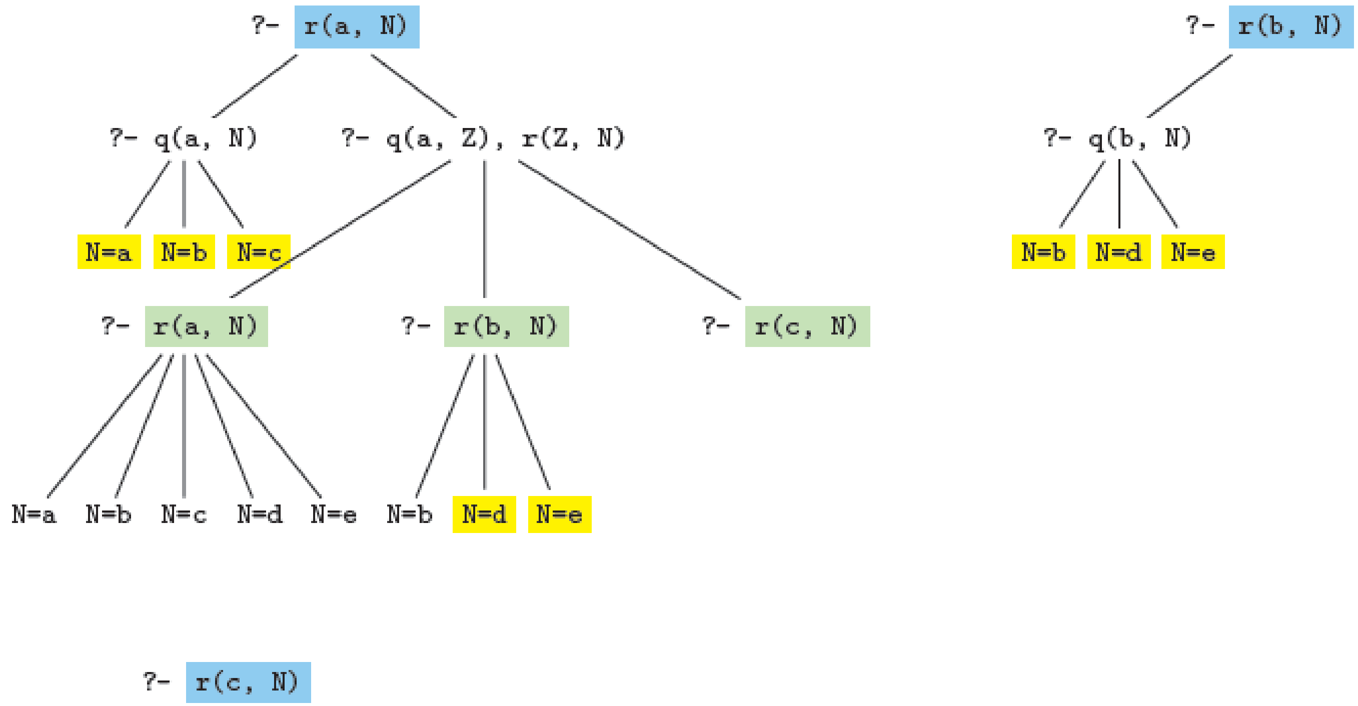





```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).


%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLDT: Second Example



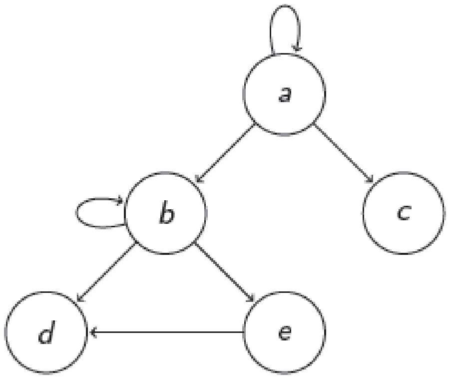



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion

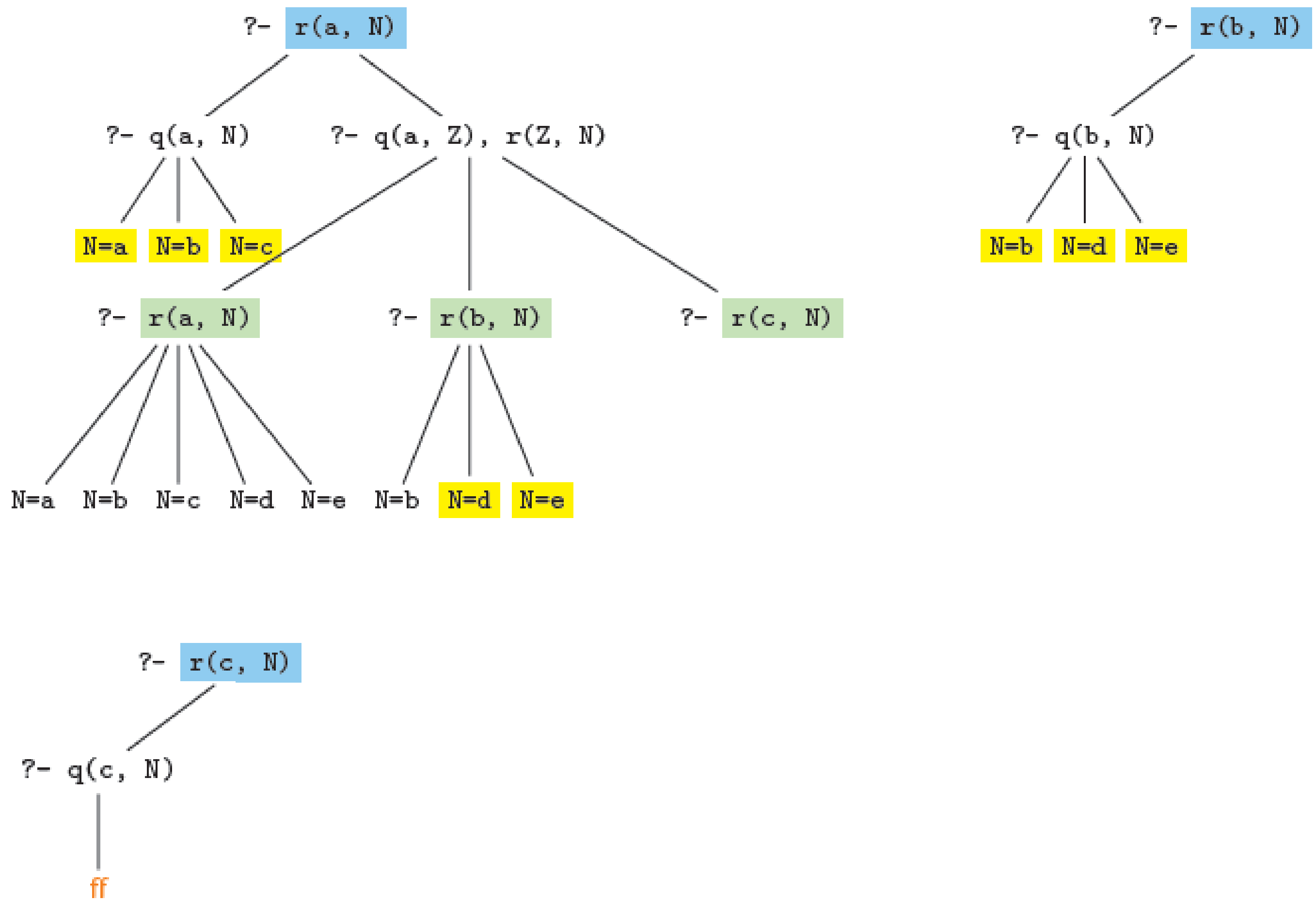
```
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLDT: Second Example

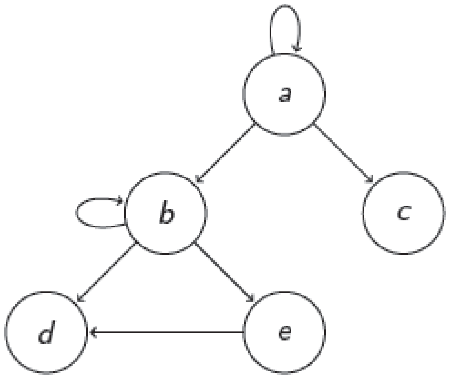



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

```
%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```



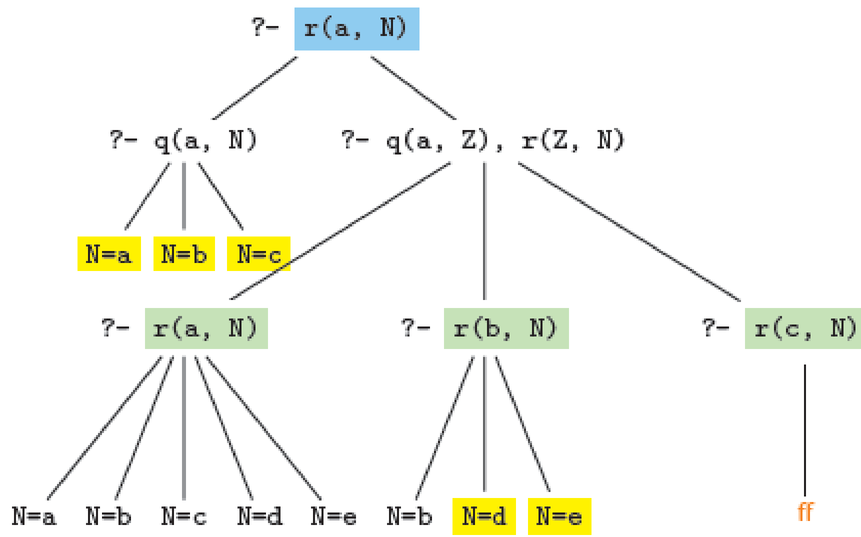
OLDT: Second Example

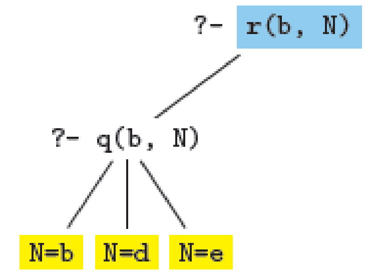


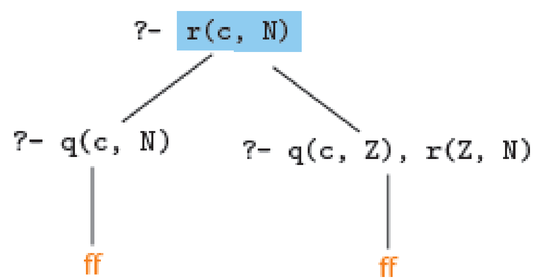

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion

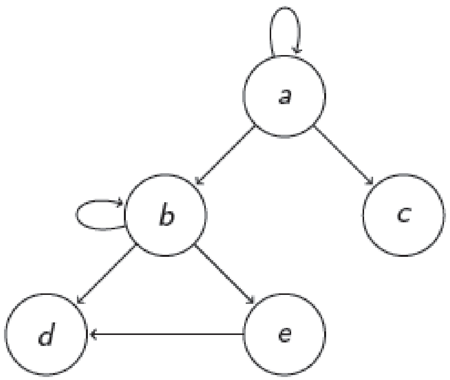
```
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

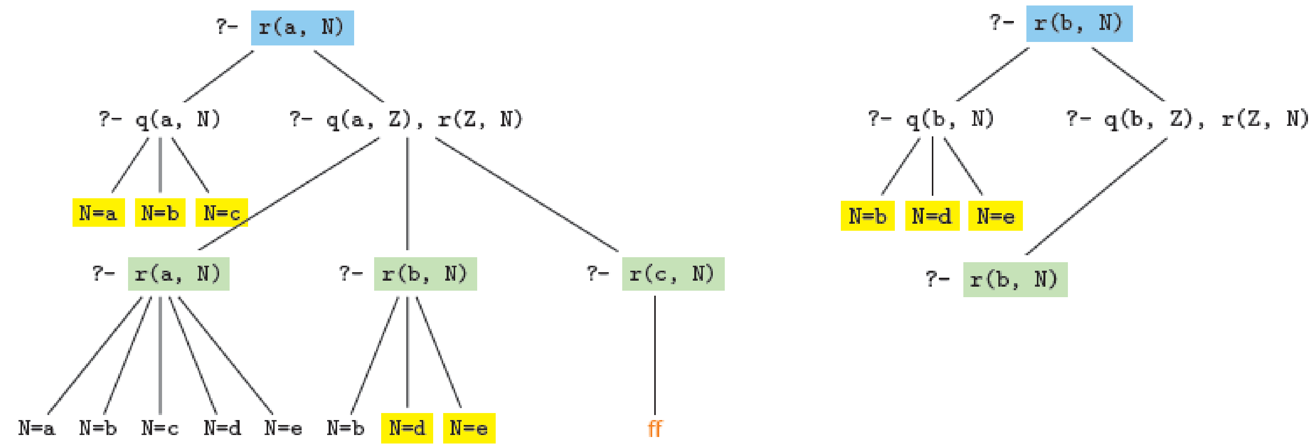




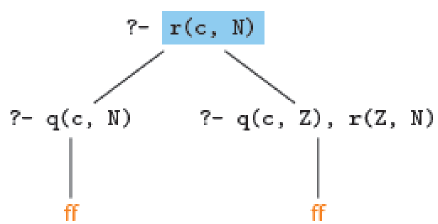


OLDT: Second Example



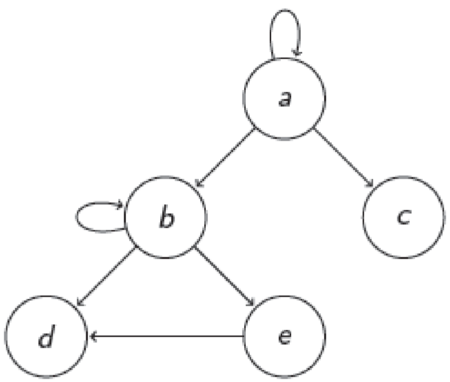


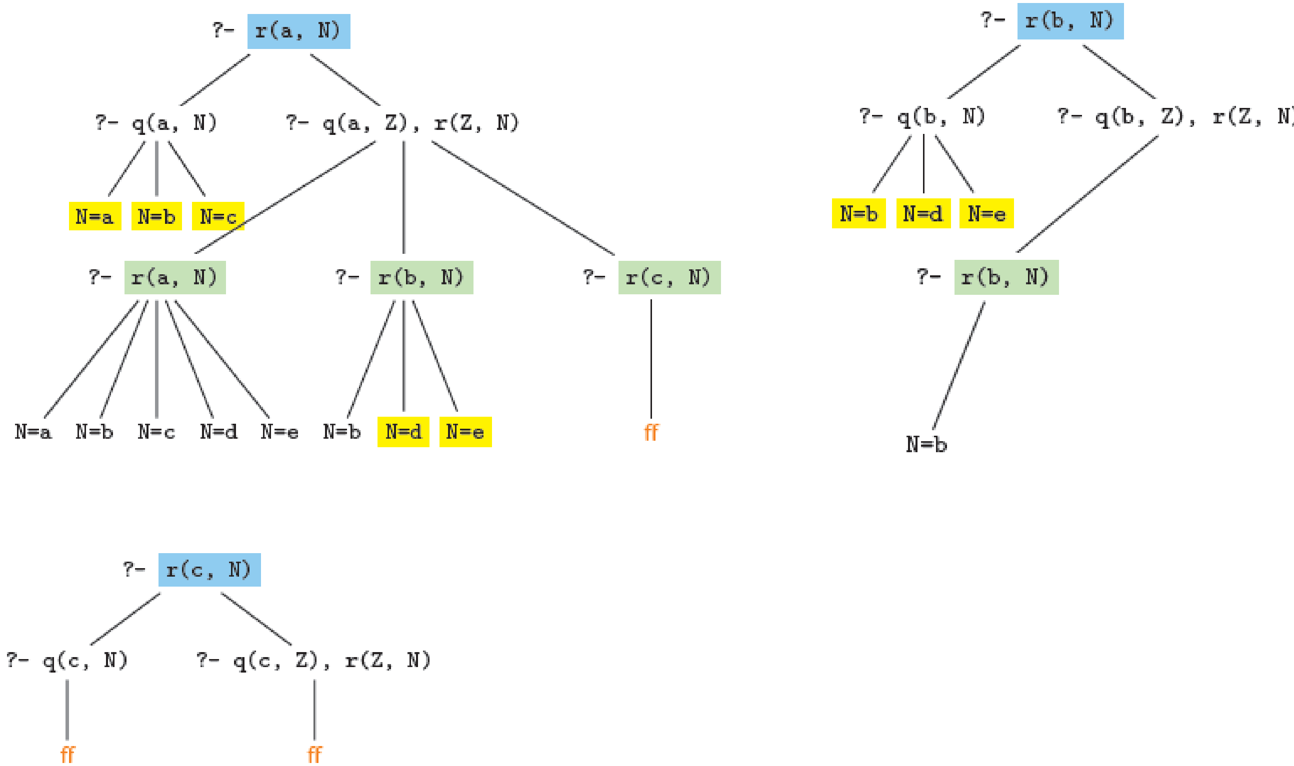

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```




```
%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLDT: Second Example



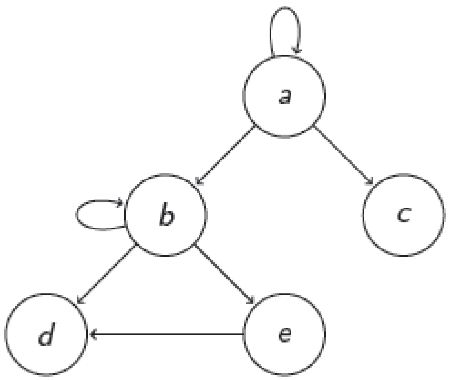



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion

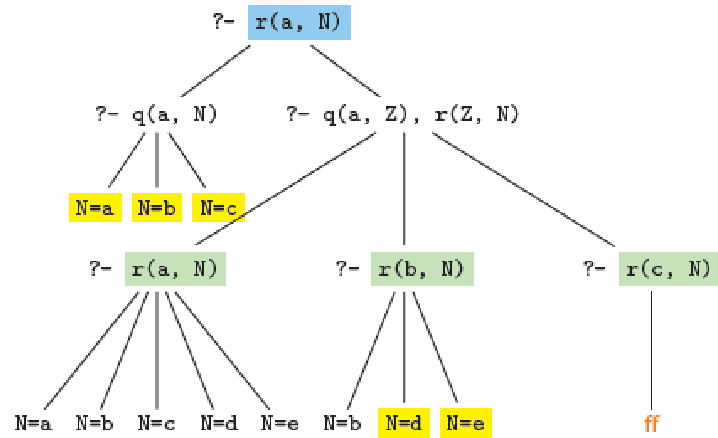
```
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

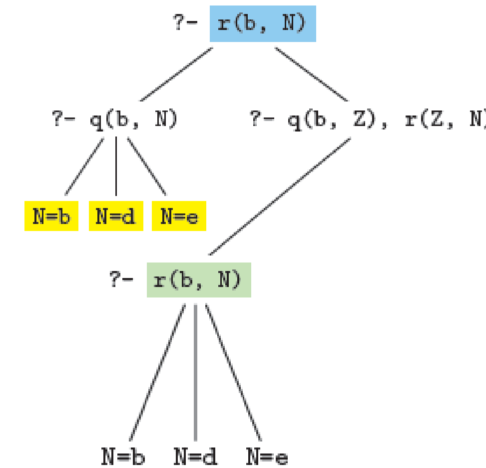
OLDT: Second Example

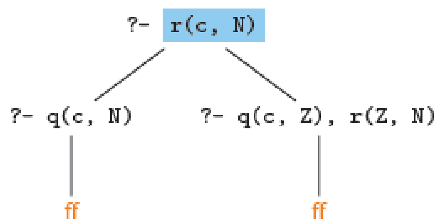



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

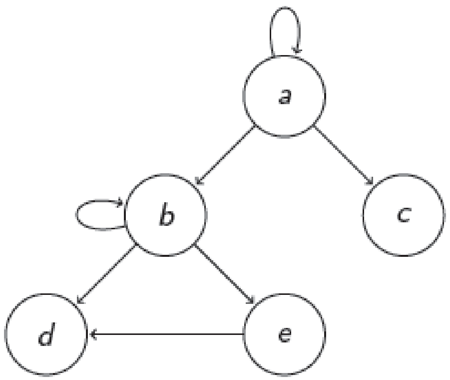
```
%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```







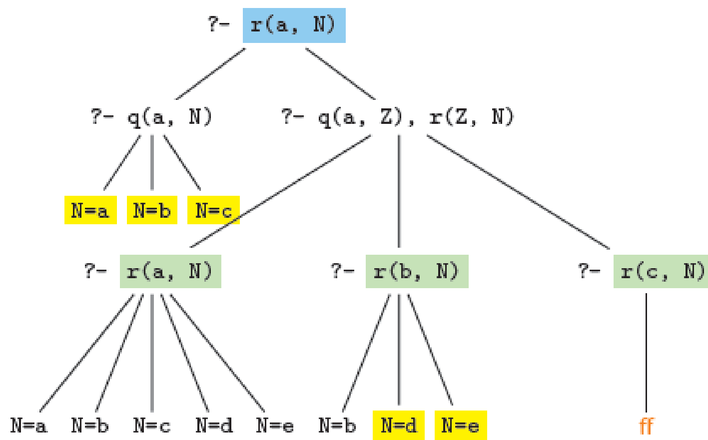
OLDT: Second Example

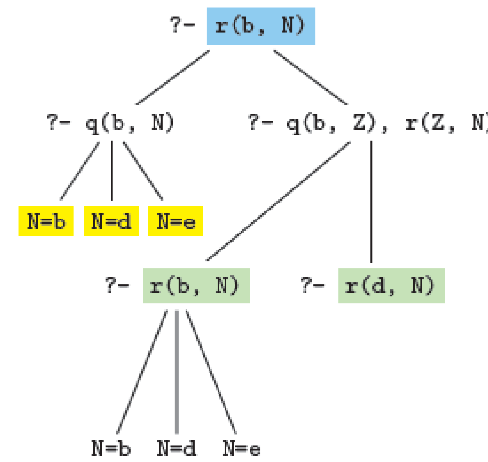


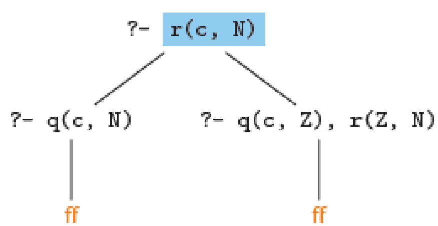

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion

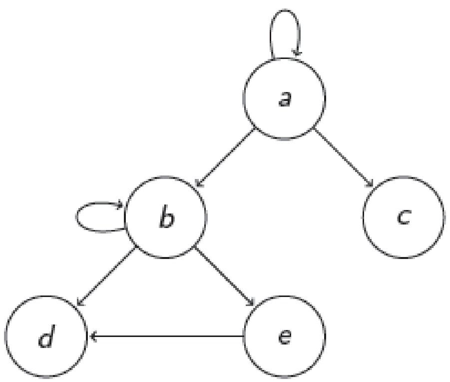
```
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

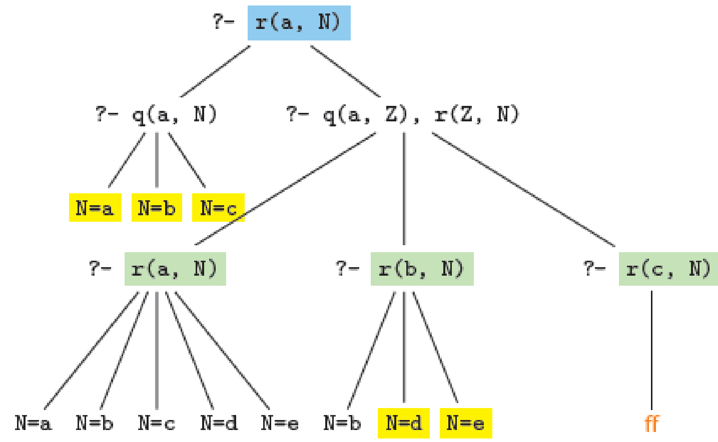


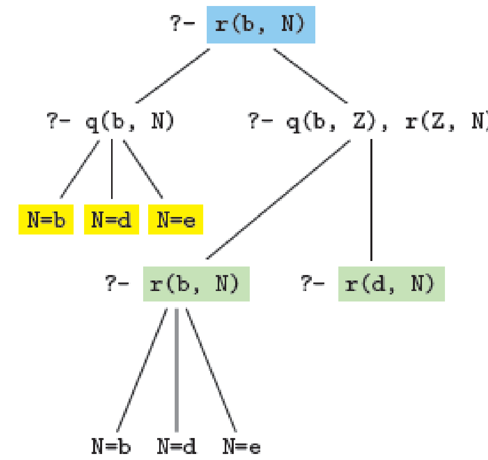




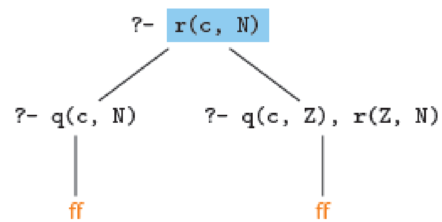
OLDT: Second Example







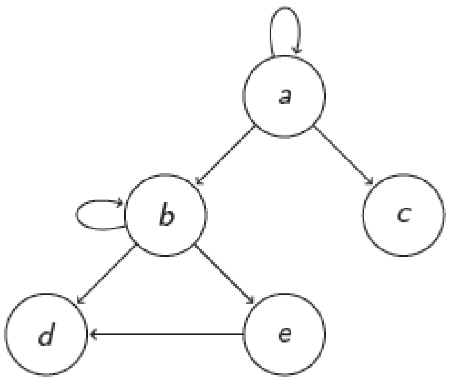

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

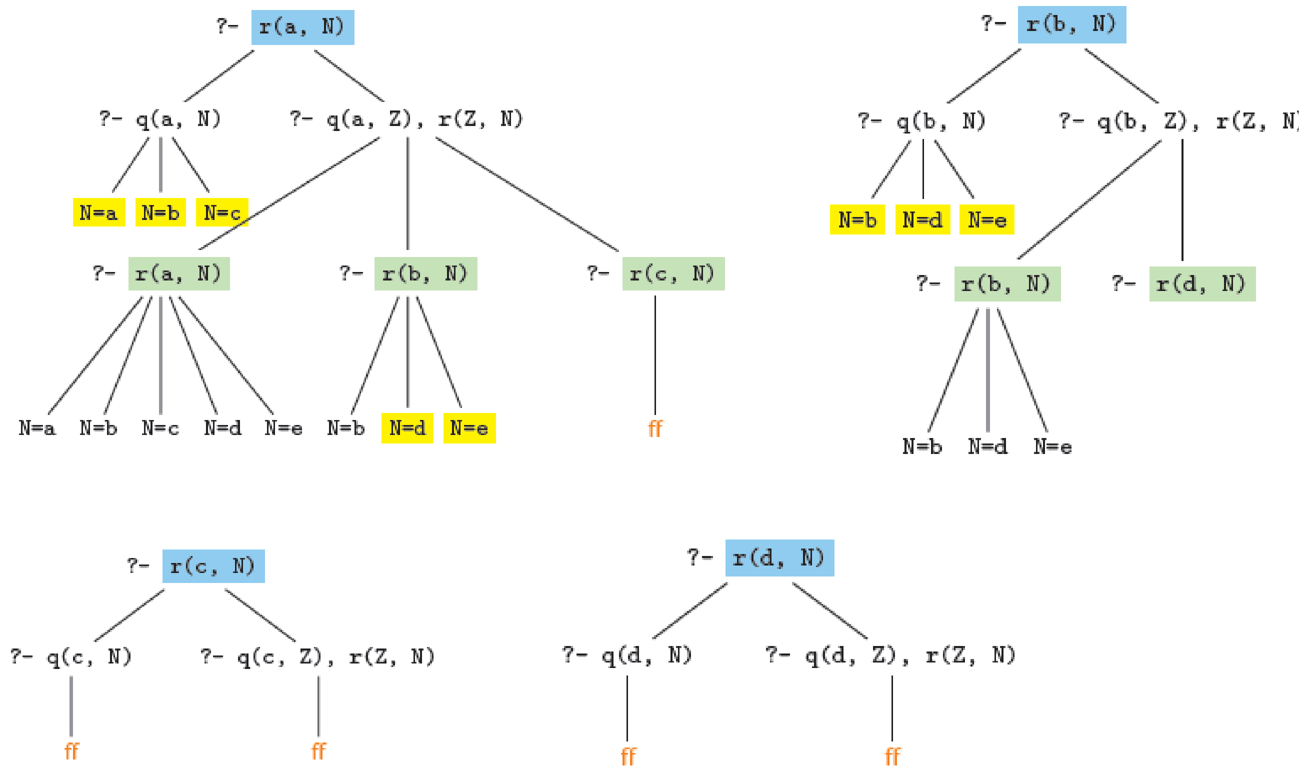


?- r(d, N)

```
%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLDT: Second Example



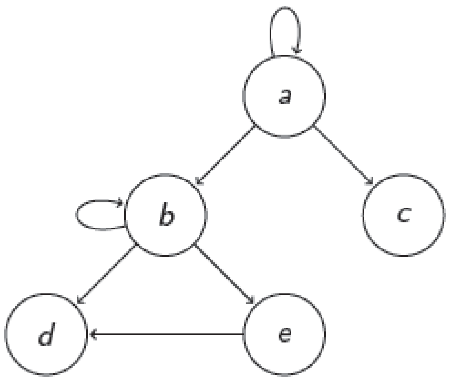



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion

```
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

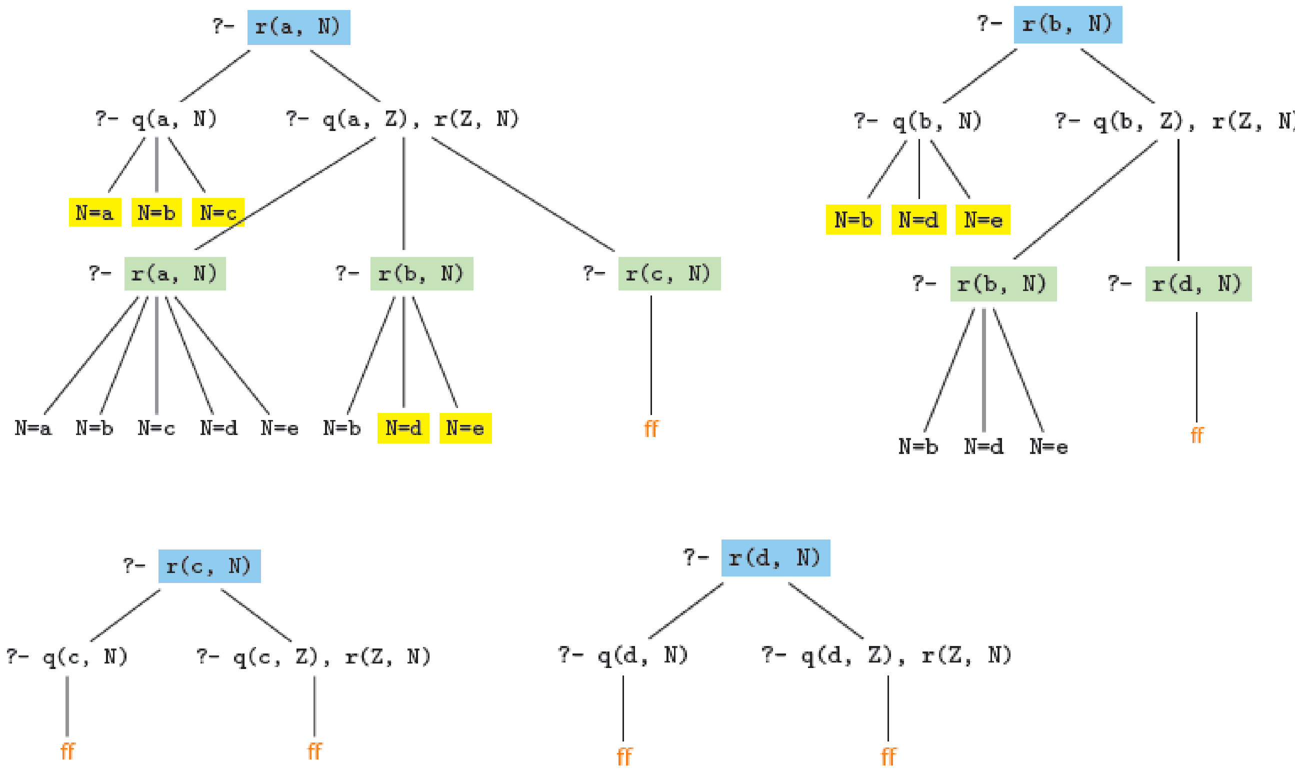
OLDT: Second Example



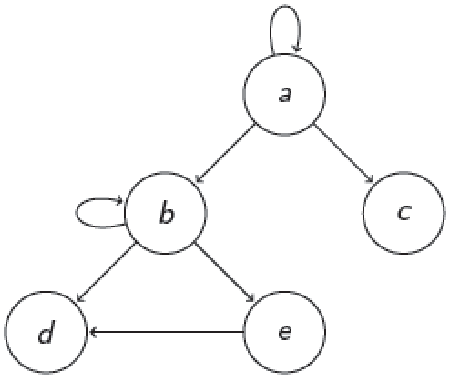

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion

```
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```



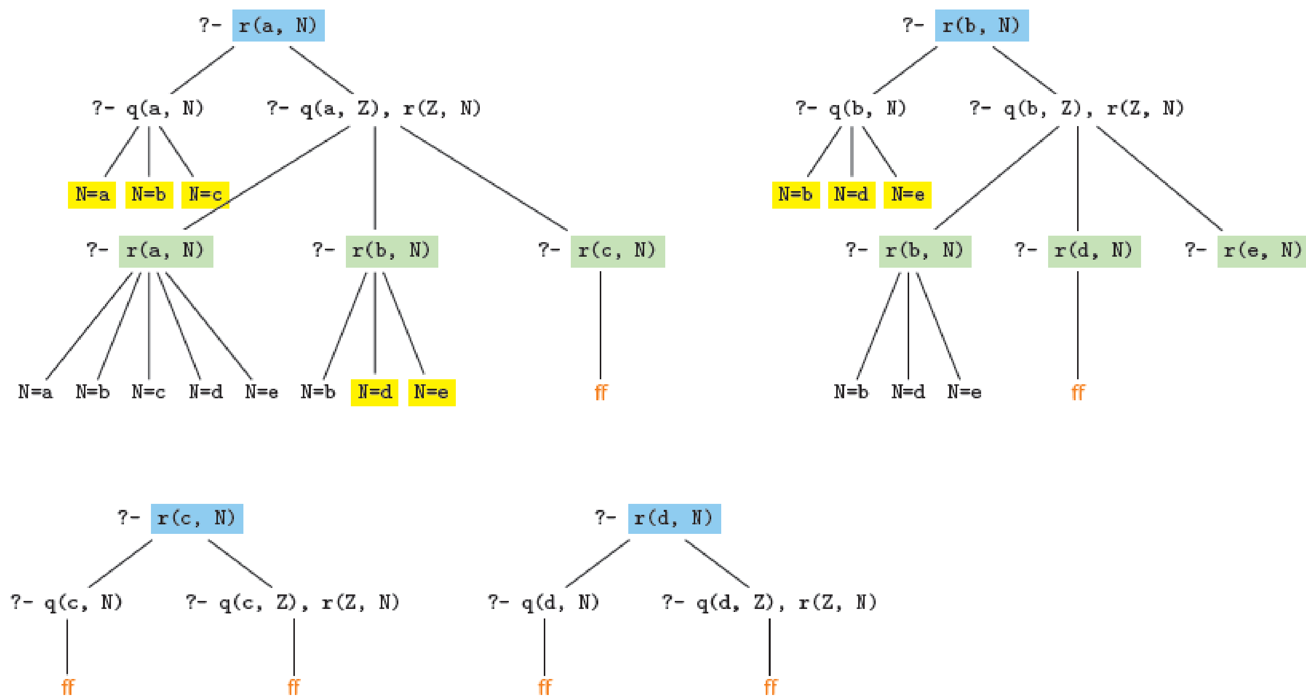
OLDT: Second Example



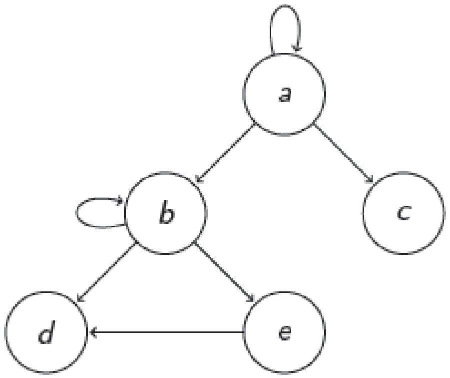

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion

```
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```



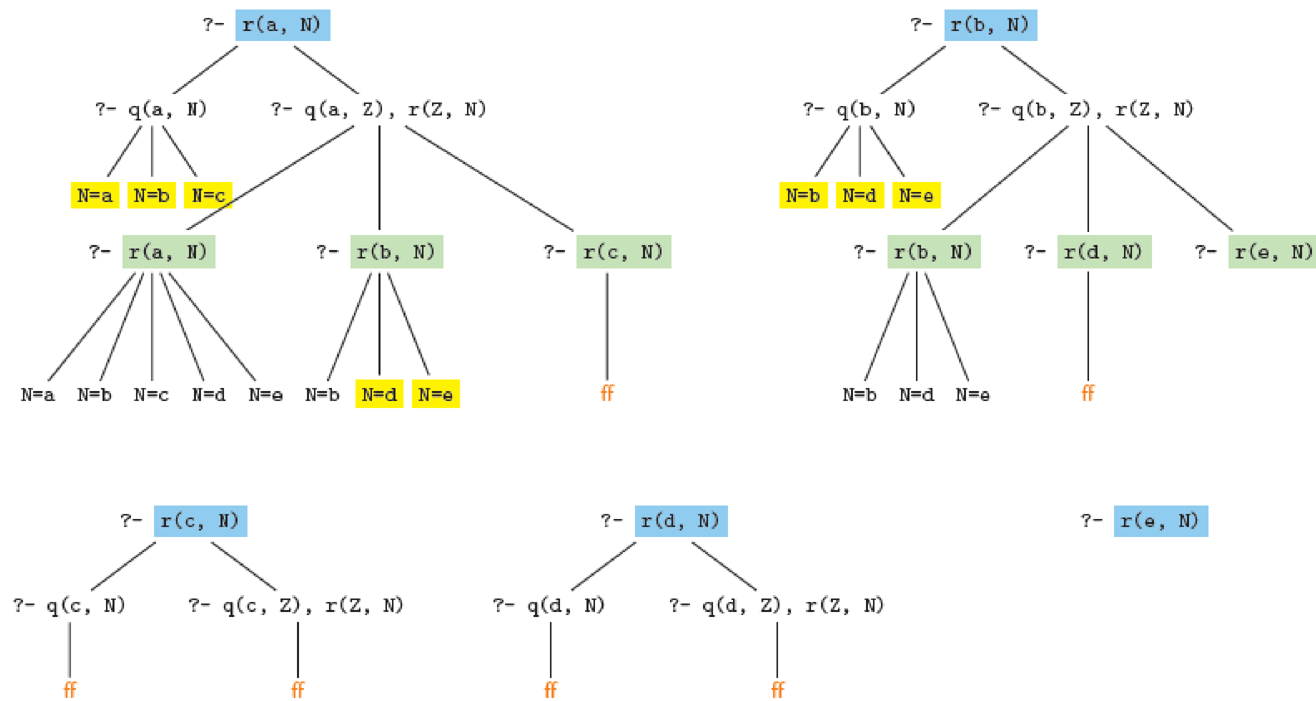
OLDT: Second Example



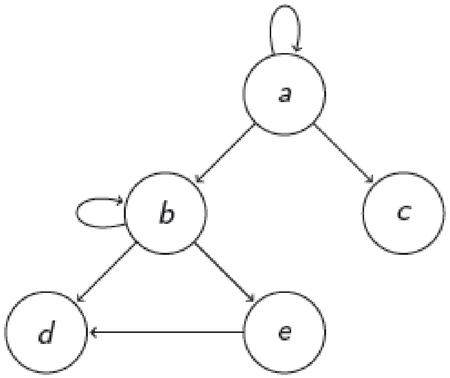

```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

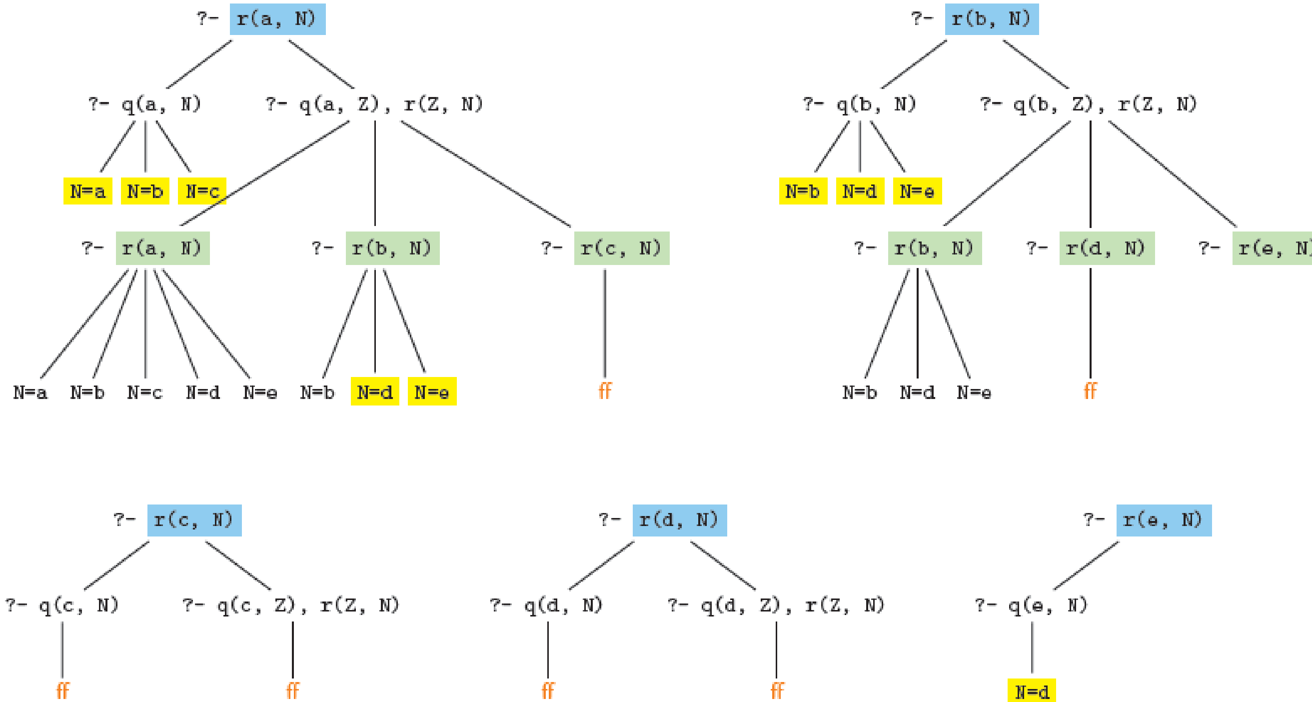
%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion

```
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```



OLDT: Second Example

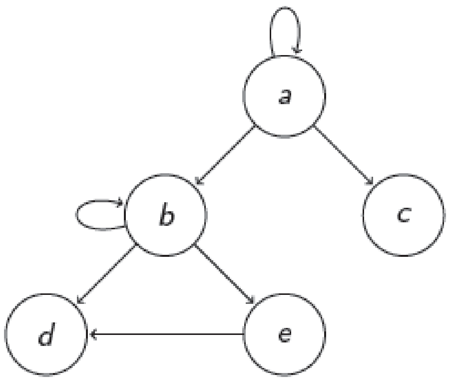


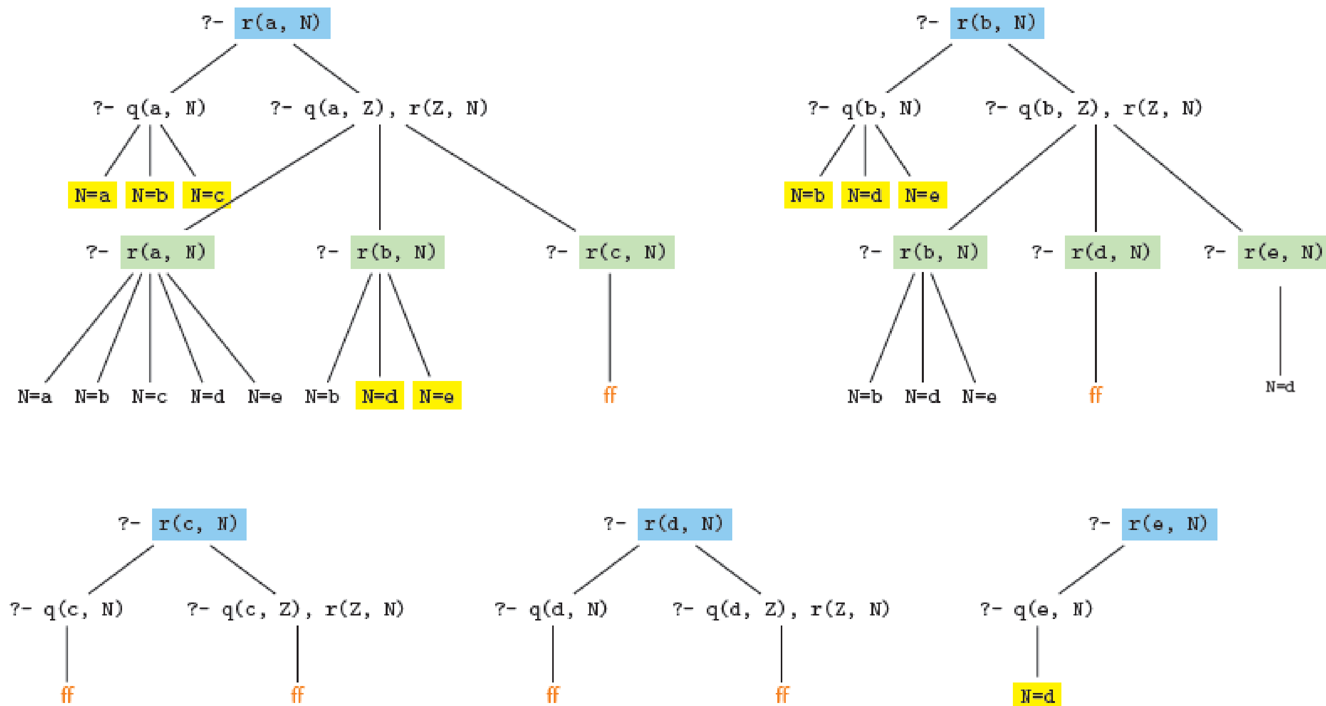



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

```
%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLDT: Second Example

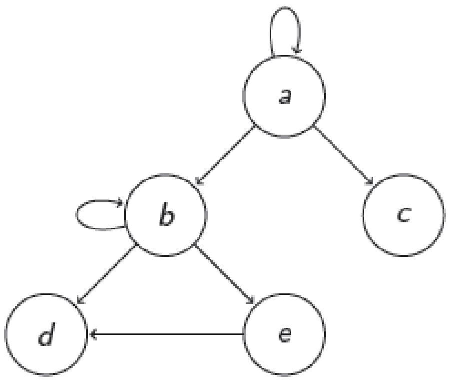


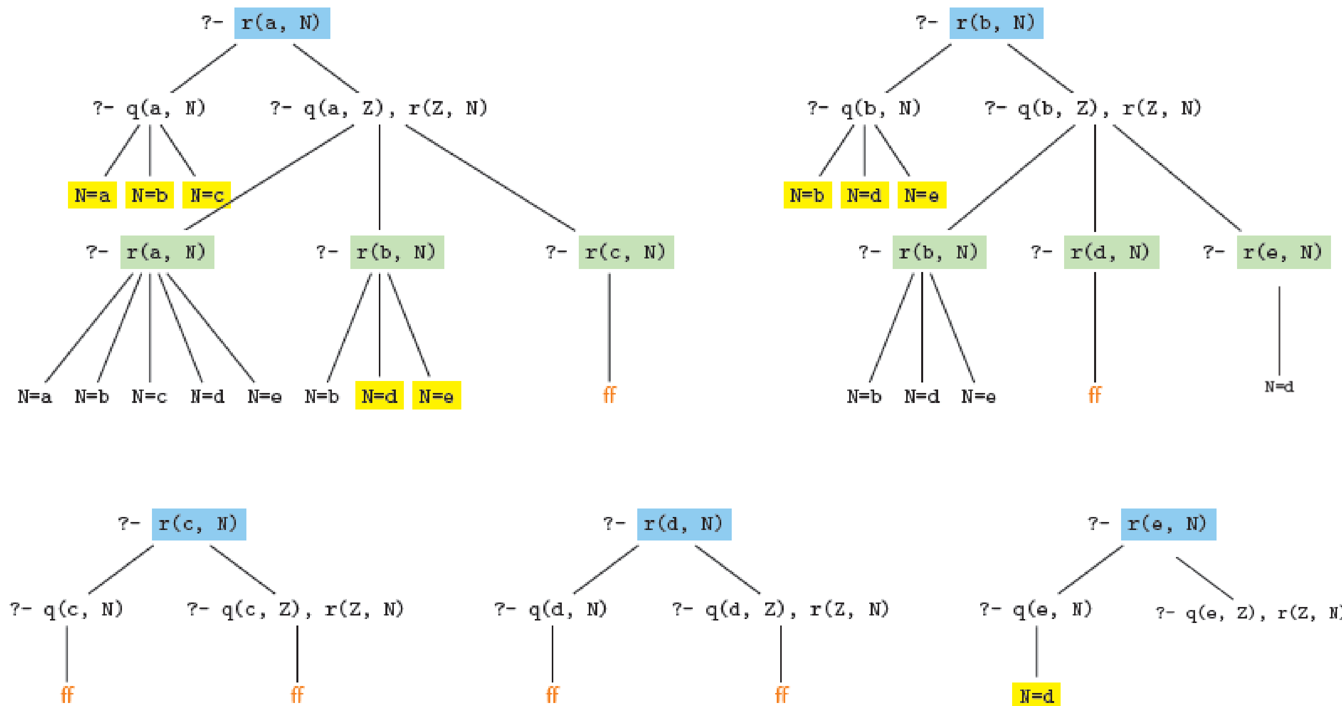



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

```
%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLDT: Second Example

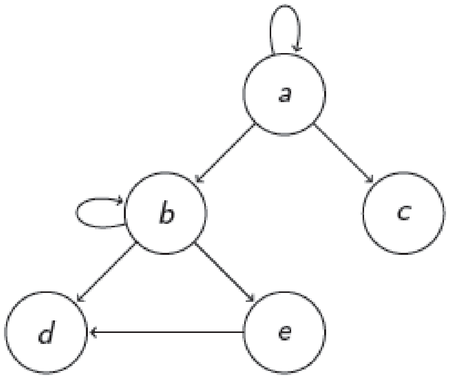


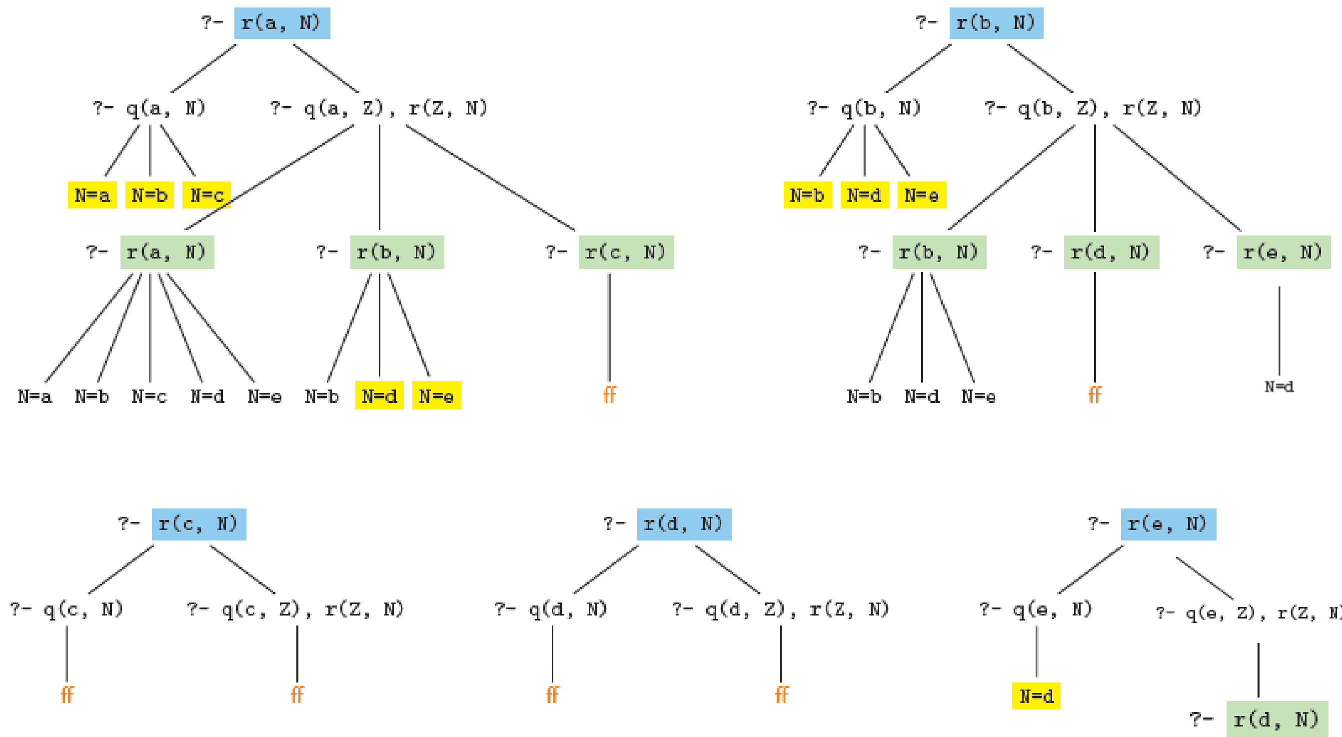



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

```
%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLDT: Second Example

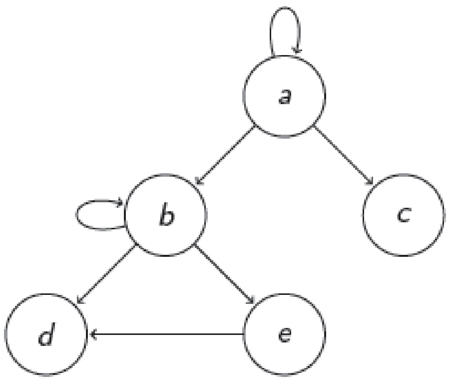


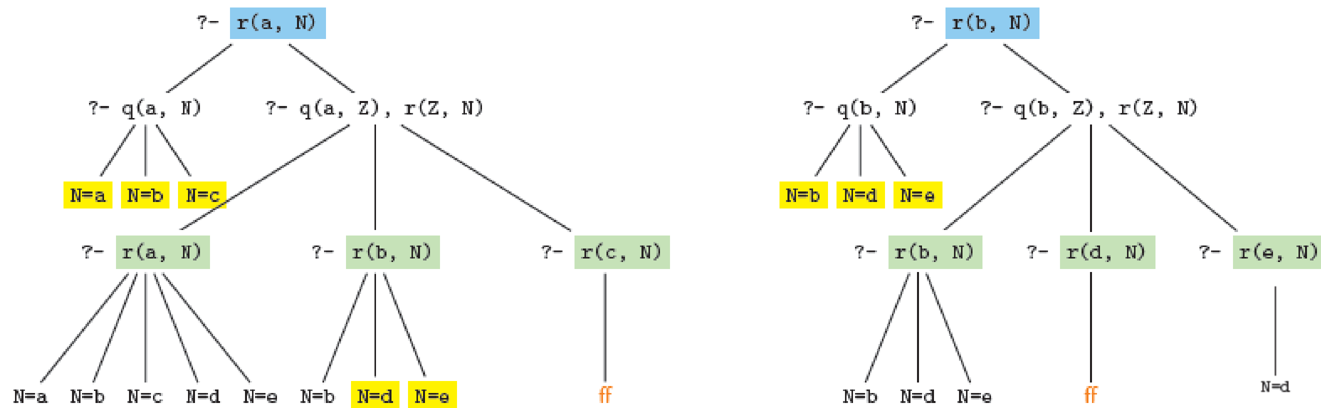



```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```

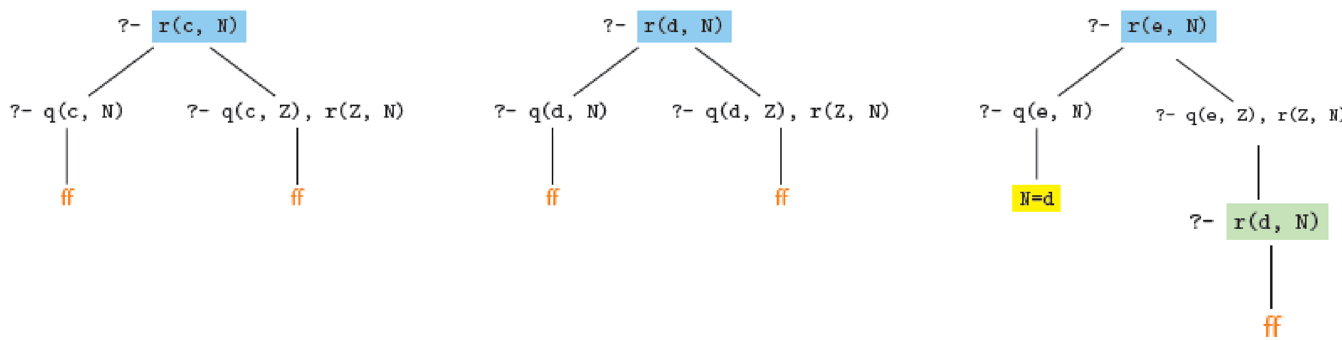
```
%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLDT: Second Example






```
q(a, a).
q(a, b).
q(a, c).
q(b, b).
q(b, d).
q(b, e).
q(e, d).
```



%ORIGINAL formulation
% of reachability: Note
% use of RIGHT recursion

```
r(X,Y) :- q(X,Y).
r(X,Y) :-
    q(X,Z), r(Z,Y).
```

OLD Resolution with Tabling (OLDT)

- OLDT evaluation can be used to infer **negative answers**: e.g. a vertex is not reachable from another
- Note that Breadth-First evaluation, or even the evaluation with goal-based stopping condition cannot do this

Tabled Resolution in XSB

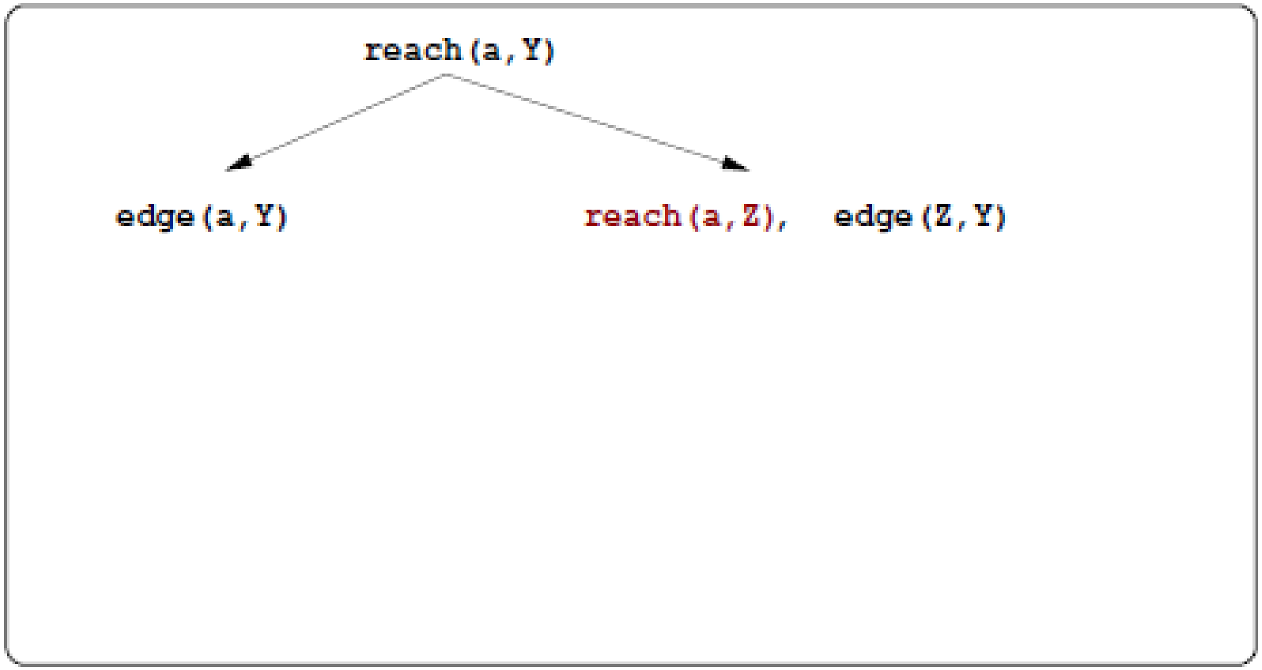
```
edge(a,a).
edge(a,b).
edge(b,c).
:- table(reach/2).
%OR :- auto_table.
reach(X,Y) :- edge(X,Y).
reach(X,Y) :- reach(X,Z), edge(Z,Y).
```

- **Call:**

```
?- reach(a,V).
```

Tabled Resolution

```
reach(X,Y) :- edge(X,Y).
reach(X,Y) :- reach(X,Z), edge(Z,Y).
edge(a,a).
edge(a,b).
edge(b,c).
```



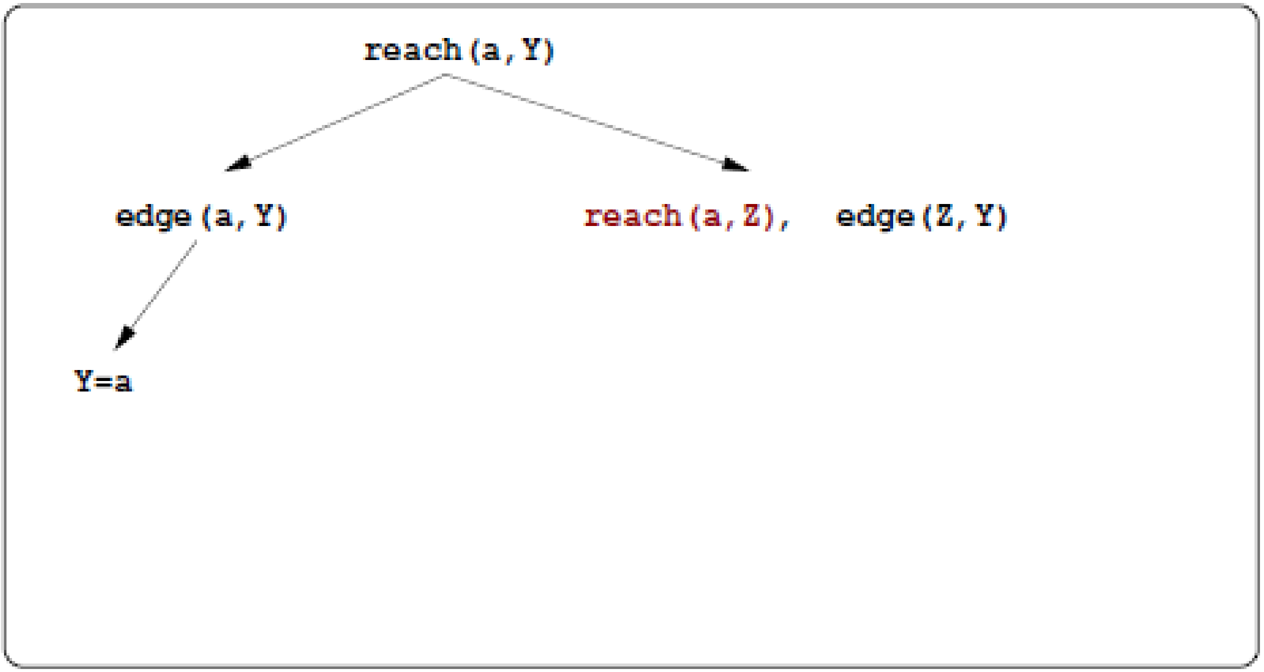
- **Calls**

```
?- reach(a,V).
```

Answers

Tabled Resolution

```
reach(X,Y) :- edge(X,Y).
reach(X,Y) :- reach(X,Z), edge(Z,Y).
edge(a,a).
edge(a,b).
edge(b,c).
```



- Calls

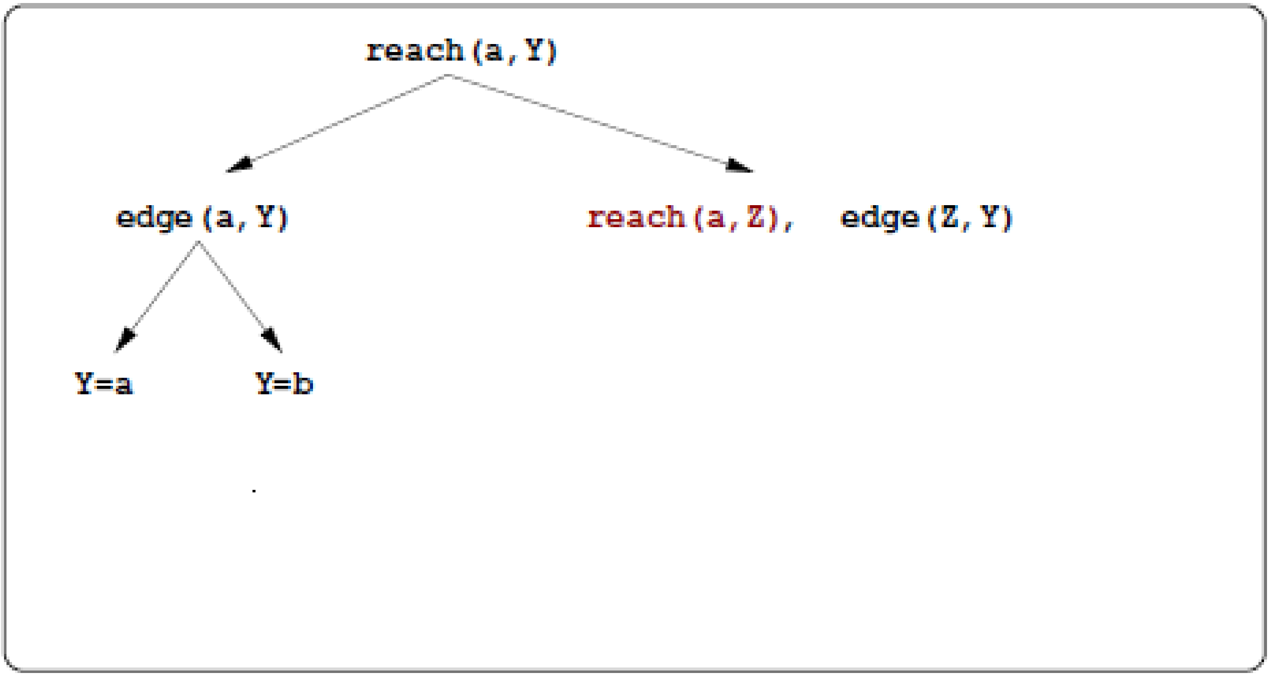
```
?- reach(a,V).
```

Answers

V = a

Tabled Resolution

```
reach(X,Y) :- edge(X,Y).
reach(X,Y) :- reach(X,Z), edge(Z,Y).
edge(a,a).
edge(a,b).
edge(b,c).
```



- Calls

```
?- reach(a,V).
```

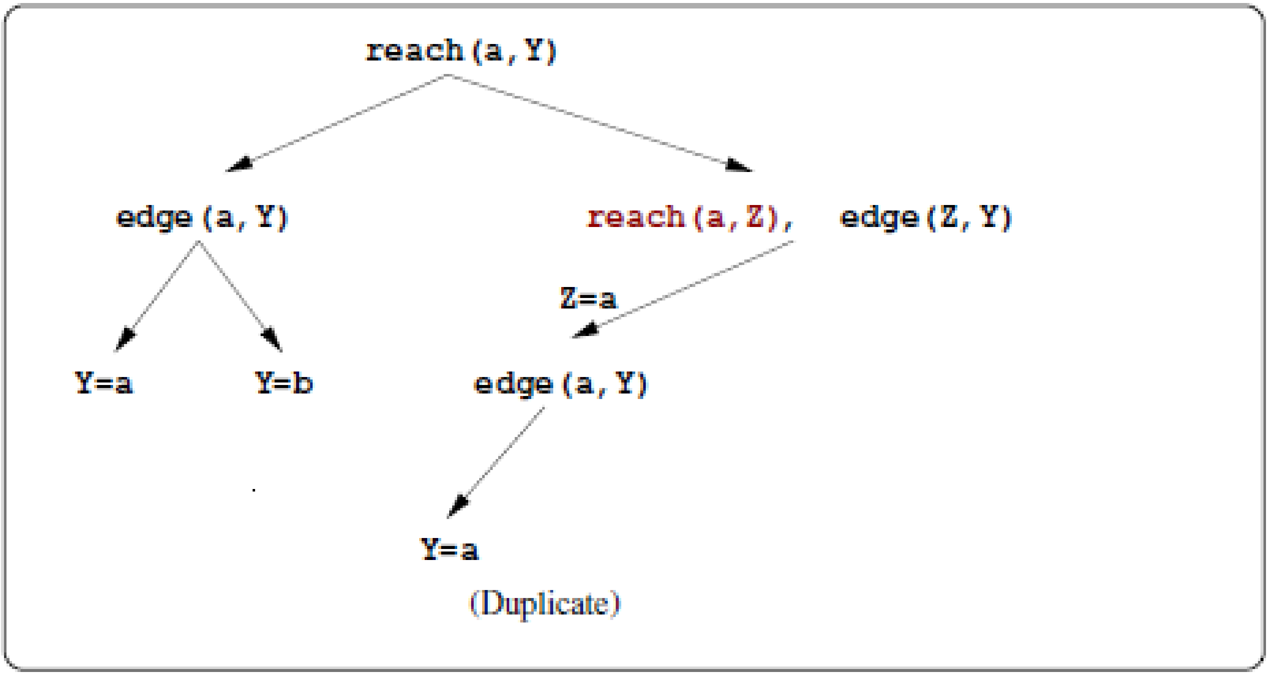
Answers

V = a

V = b

Tabled Resolution

```
reach(X,Y) :- edge(X,Y).
reach(X,Y) :- reach(X,Z), edge(Z,Y).
edge(a,a).
edge(a,b).
edge(b,c).
```



- Calls

```
?- reach(a,V).
```

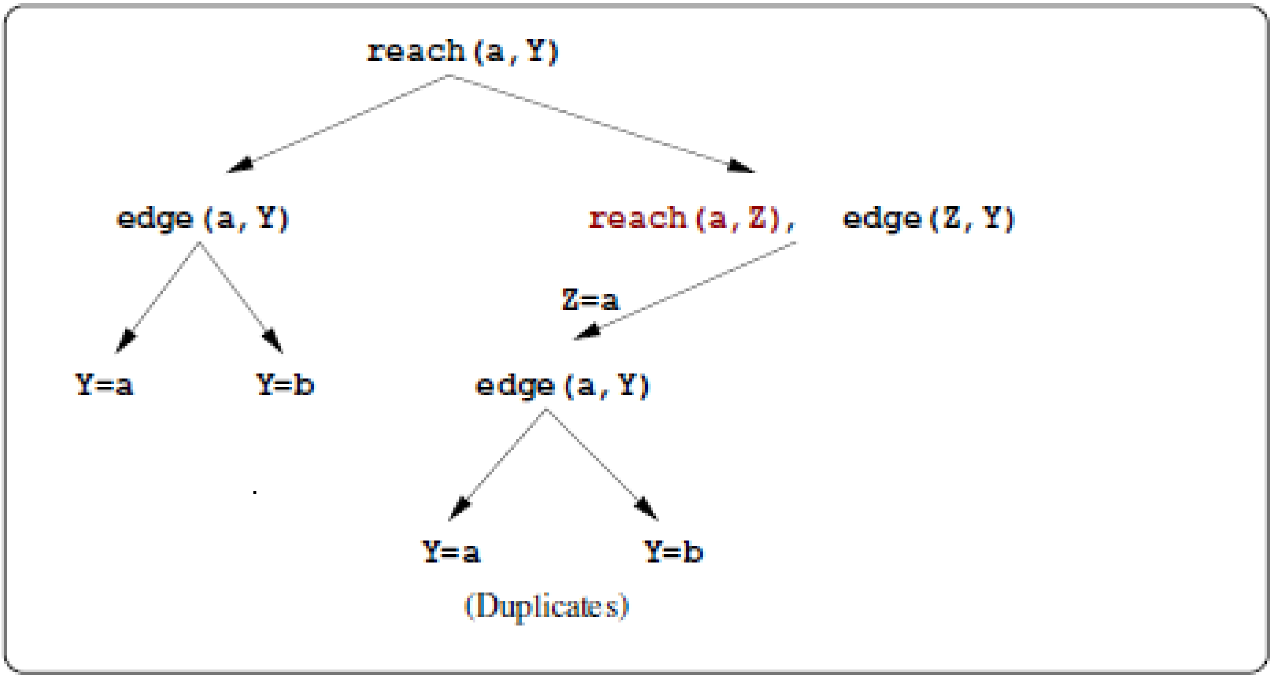
Answers

V = a

V = b

Tabled Resolution

```
reach(X,Y) :- edge(X,Y).
reach(X,Y) :- reach(X,Z), edge(Z,Y).
edge(a,a).
edge(a,b).
edge(b,c).
```



- Calls

```
?- reach(a,V).
```

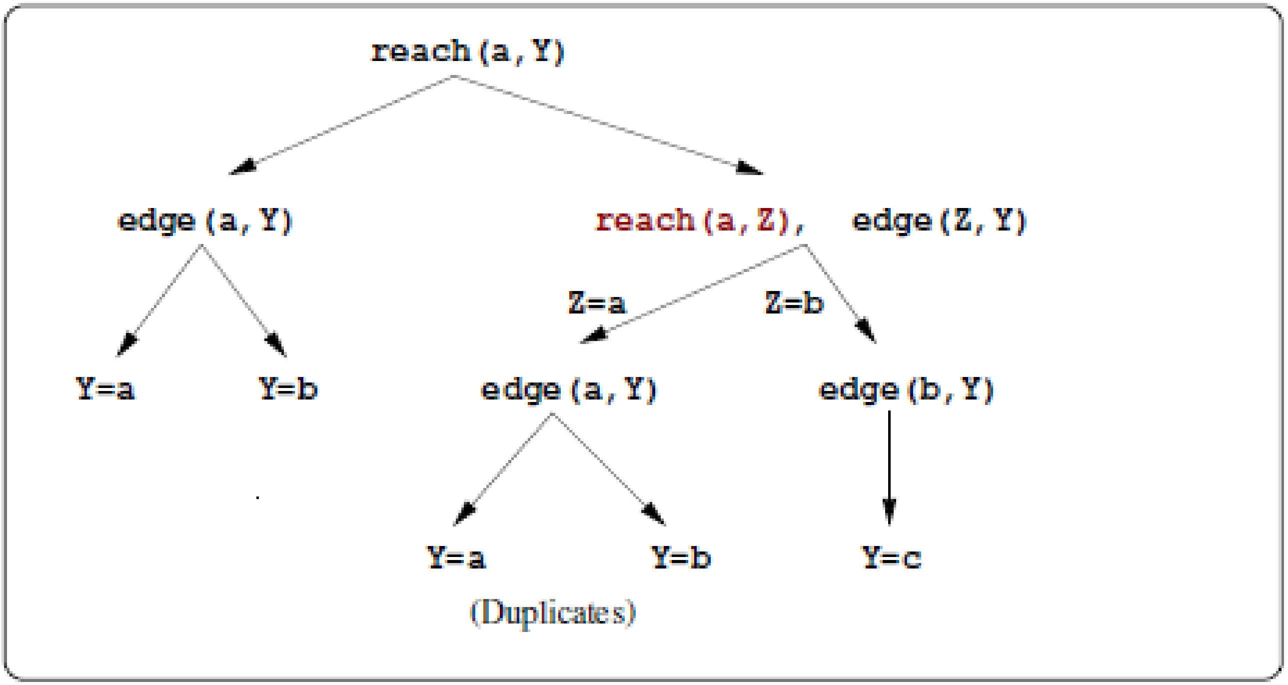
Answers

V = a

V = b

Tabled Resolution

```
reach(X,Y) :- edge(X,Y).
reach(X,Y) :- reach(X,Z), edge(Z,Y).
edge(a,a).
edge(a,b).
edge(b,c).
```



- Calls

```
?- reach(a,V).
```

Answers

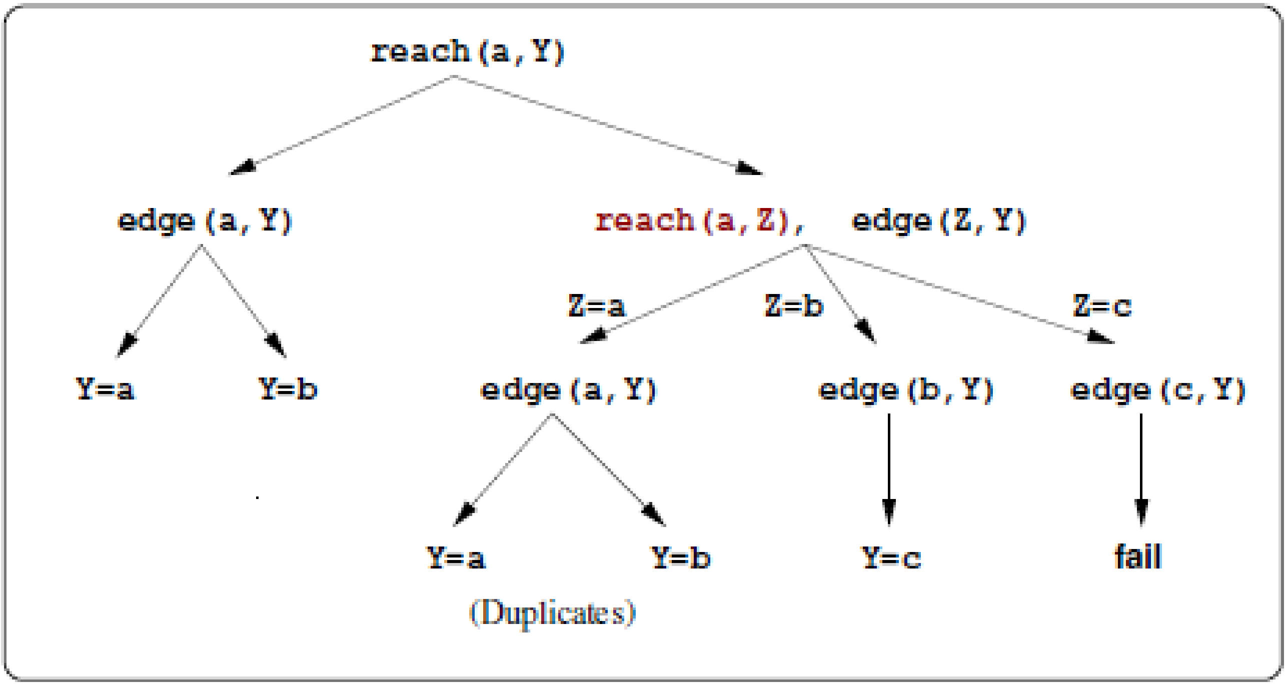
V = a

V = b

V = c

Tabled Resolution

```
reach(X,Y) :- edge(X,Y).
reach(X,Y) :- reach(X,Z), edge(Z,Y).
edge(a,a).
edge(a,b).
edge(b,c).
```



- Calls

```
?- reach(a,V).
```

Answers

V = a

V = b

V = c

Answer completion!