Esercizio

Si consideri il seguente programma Prolog:

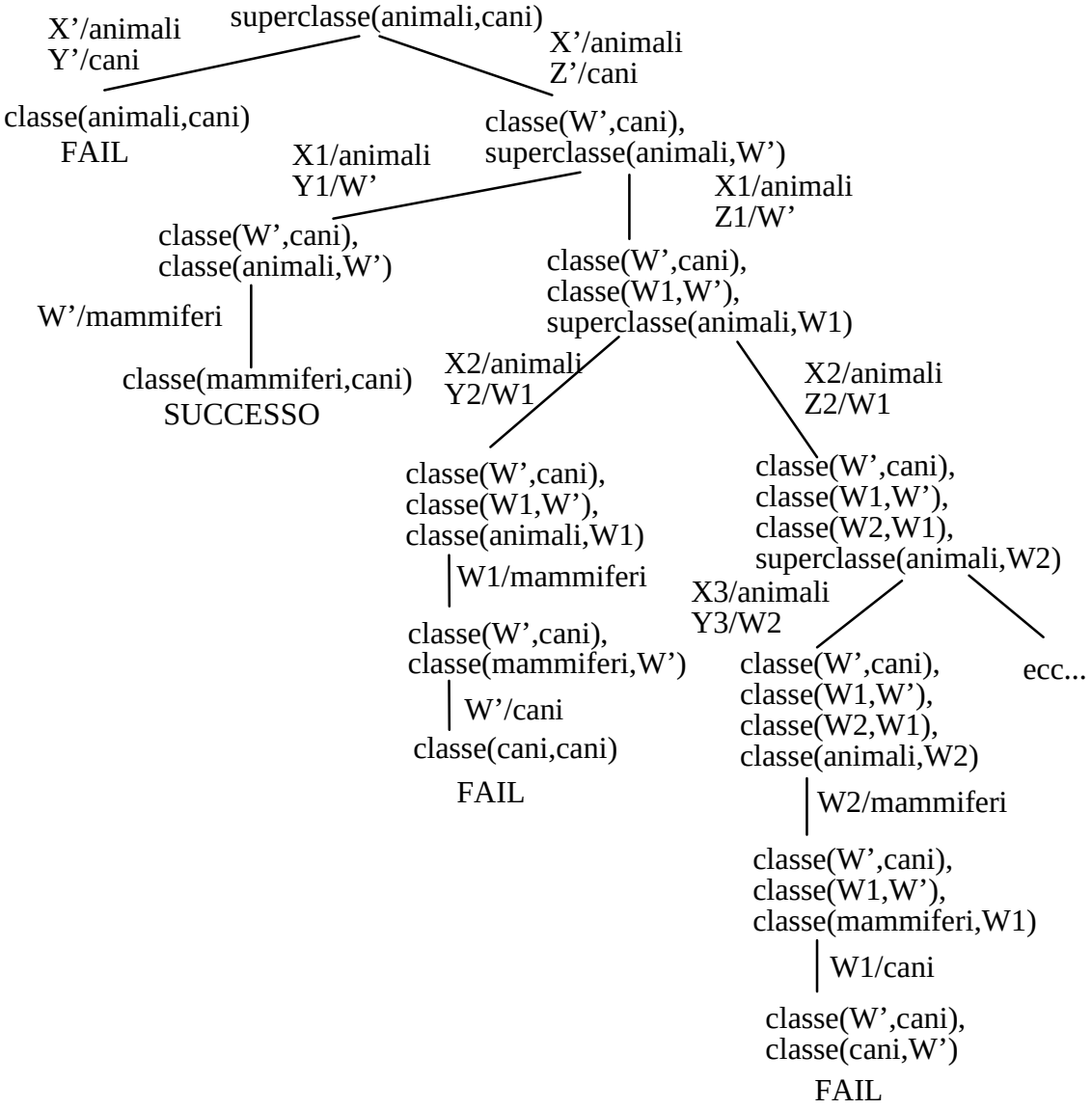
```
superclasse(X,Y):- classe(X,Y).
superclasse(X,Z):- classe(W,Z),
                   superclasse(X,W).
```

Si rappresenti l'albero SLD con regola di selezione **right-most** relativo al goal:

```
:- superclasse(animali, cani)
```

assumendo la presenza, all'inizio del database, dei fatti:

```
classe(mammiferi,cani).
classe(animali,mammiferi).
```


superclasse(animali,cani)
X'/animali
Y'/cani
X'/animali
Z'/cani
classe(animali,cani)
FAIL
classe(W',cani),
superclasse(animali,W')
X1/animali
Y1/W'
X1/animali
Z1/W'
classe(W',cani),
classe(animali,W')
classe(W',cani),
classe(W1,W'),
superclasse(animali,W1)
W'/mammiferi
classe(mammiferi,cani)
SUCCESSO
X2/animali
Y2/W1
X2/animali
Z2/W1
classe(W',cani),
classe(W1,W'),
classe(animali,W1)
classe(W',cani),
classe(W1,W'),
classe(W2,W1),
superclasse(animali,W2)
W1/mammiferi
X3/animali
Y3/W2
classe(W',cani),
classe(mammiferi,W')
classe(W',cani),
classe(W1,W'),
classe(W2,W1),
classe(animali,W2)
ecc...
W'/cani
classe(cani,cani)
FAIL
W2/mammiferi
classe(W',cani),
classe(W1,W'),
classe(mammiferi,W1)
W1/cani
classe(W',cani),
classe(cani,W')
FAIL

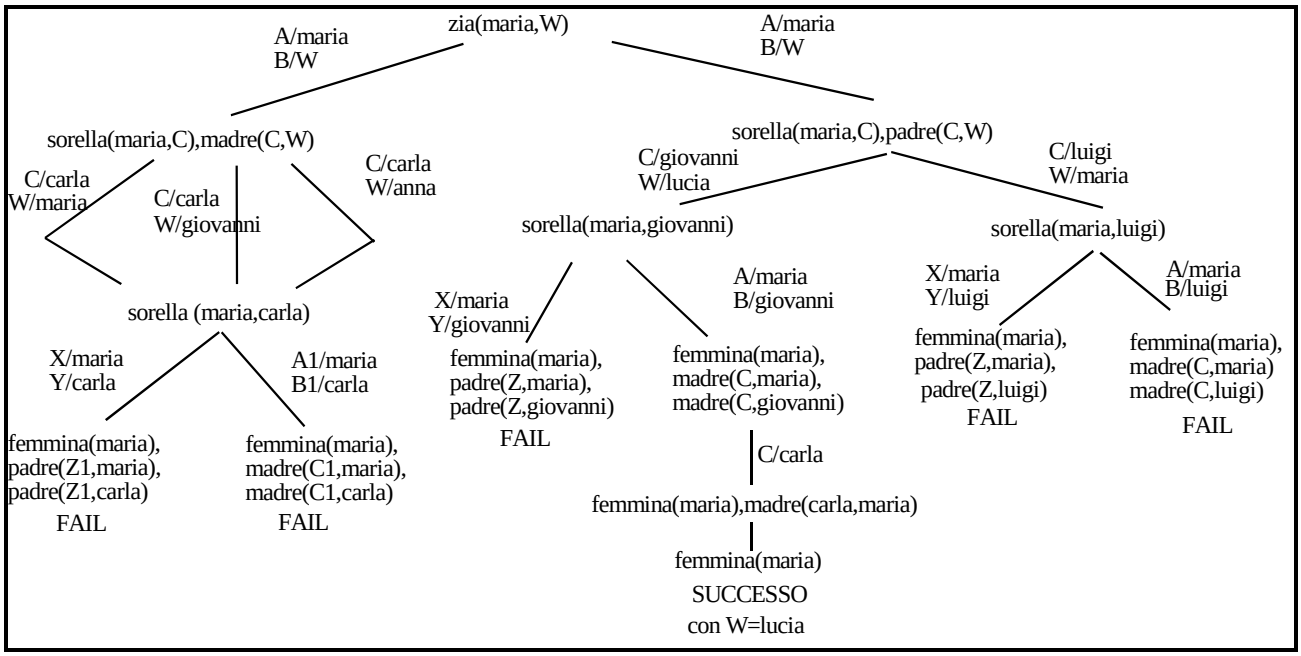
Esercizio

Dati i seguenti fatti e regole:

```
femmina(carla).
femmina(maria).
femmina(anna).
femmina(maria).
femmina(lucia).
madre(carla,maria).
madre(carla,giovanni).
madre(carla,anna).
padre(giovanni,lucia).
padre(luigi,maria).
sorella(X,Y):-     femmina(X),
                   padre(Z,X),
                   padre(Z,Y).
sorella(A,B):-     femmina(A),
                   madre(C,A),
                   madre(C,B).
zia(A,B) :- sorella(A,C), madre(C,B).
zia(A,B) :- sorella(A,C), padre(C,B).
```

ed il goal: `zia(maria,W).`

Si mostri l'albero SLD generato in caso di regola di computazione **right-most**.


zia(maria,W)
A/maria
B/W
A/maria
B/W
sorella(maria,C),madre(C,W)
C/carla
W/maria
C/carla
W/giovanni
C/carla
W/anna
sorella (maria,carla)
X/maria
Y/carla
A1/maria
B1/carla
femmina(maria),
padre(Z1,maria),
padre(Z1,carla)
FAIL
femmina(maria),
madre(C1,maria),
madre(C1,carla)
FAIL
sorella(maria,C),padre(C,W)
C/giovanni
W/lucia
C/luigi
W/maria
sorella(maria,giovanni)
sorella(maria,luigi)
X/maria
Y/giovanni
A/maria
B/giovanni
X/maria
Y/luigi
A/maria
B/luigi
femmina(maria),
padre(Z,maria),
padre(Z,giovanni)
FAIL
femmina(maria),
madre(C,maria),
madre(C,giovanni)
femmina(maria),
padre(Z,maria),
padre(Z,luigi)
FAIL
femmina(maria),
madre(C,maria)
madre(C,luigi)
FAIL
C/carla
femmina(maria),madre(carla,maria)
femmina(maria)
SUCCESSO
con W=lucia

Esercizio

Si consideri il programma:

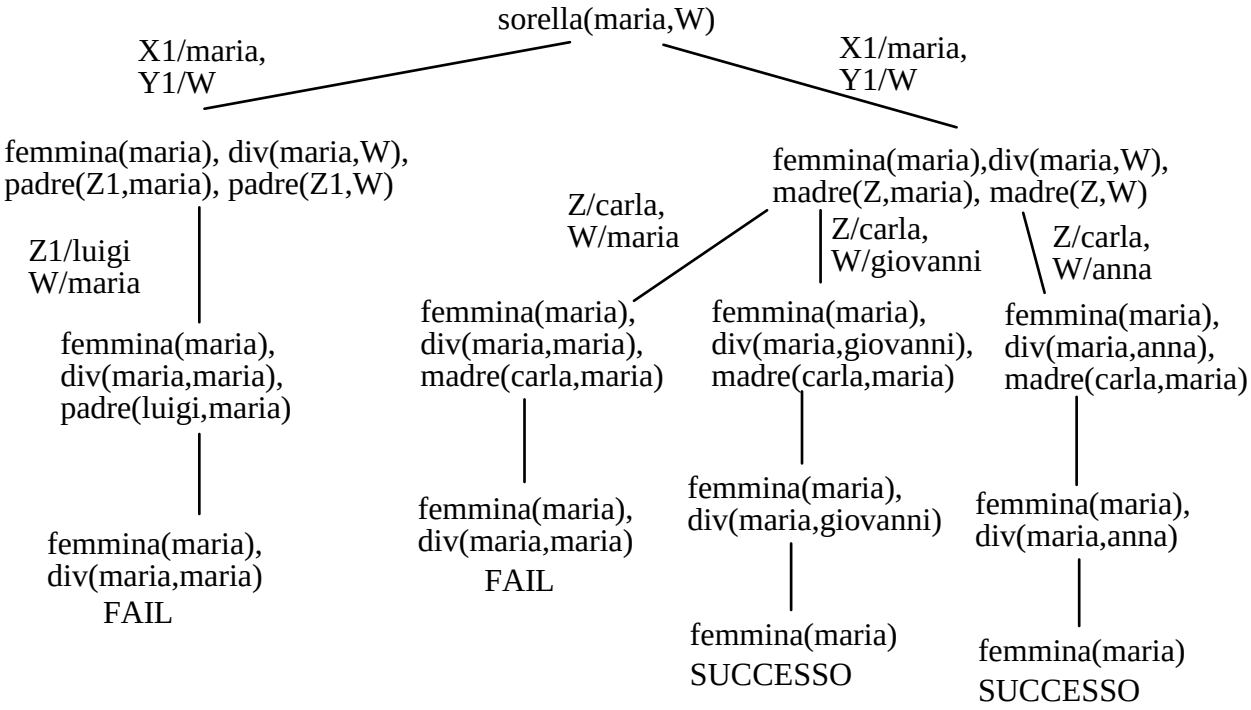
```
femmina(carla).
femmina(maria).
femmina(anna).
madre(carla,maria).
madre(carla,giovanni).
madre(carla,anna).
padre(luigi,maria).
sorella(X,Y):- femmina(X),
               div(X,Y).
               padre(Z,X),
               padre(Z,Y).
sorella(X,Y):-femmina(X),
               div(X,Y).
               madre(Z,X),
               madre(Z,Y).
div(carla,maria).
div(maria,carla).
```

..... (`div(A,B)` per tutte le coppie `(A,B)` con `A ≠ B`)

E la "query":

```
?-  sorella(maria,W).
```

Si mostri l'albero SLD generato per risolvere il goal, usando una regola di computazione **right-most**.


sorella(maria,W)
X1/maria,
Y1/W
X1/maria,
Y1/W
femmina(maria), div(maria,W),
padre(Z1,maria), padre(Z1,W)
femmina(maria),div(maria,W),
madre(Z,maria), madre(Z,W)
Z1/luigi
W/maria
Z/carla,
W/maria
Z/carla,
W/giovanni
Z/carla,
W/anna
femmina(maria),
div(maria,maria),
padre(luigi,maria)
femmina(maria),
div(maria,maria),
madre(carla,maria)
femmina(maria),
div(maria,giovanni),
madre(carla,maria)
femmina(maria),
div(maria,anna),
madre(carla,maria)
femmina(maria),
div(maria,maria)
FAIL
femmina(maria),
div(maria,maria)
FAIL
femmina(maria),
div(maria,giovanni)
femmina(maria),
div(maria,anna)
femmina(maria)
SUCCESSO
femmina(maria)
SUCCESSO

Esercizio

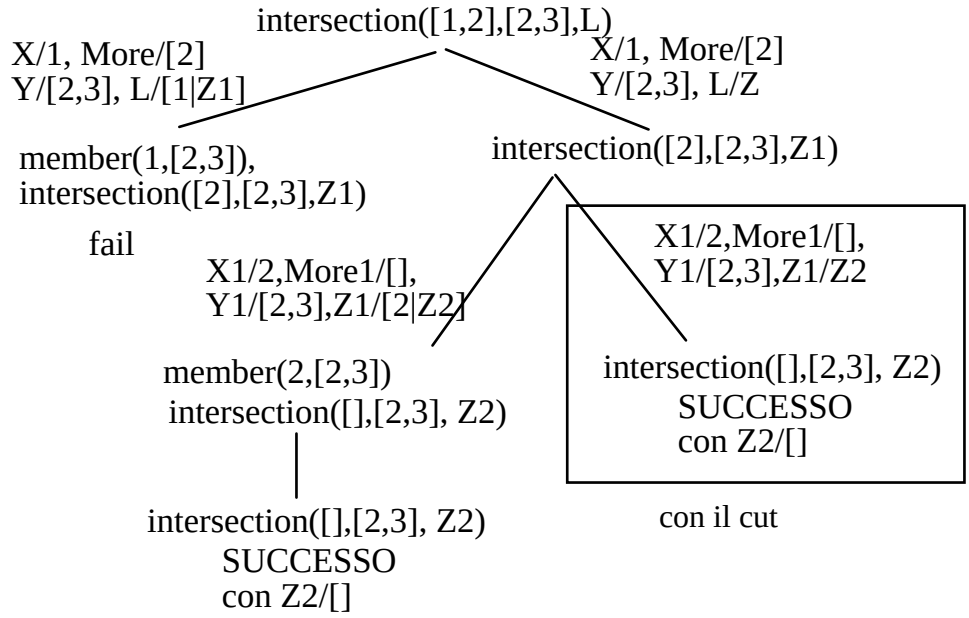
Si consideri il seguente programma Prolog:

```
intersection([],X,[]).
intersection([X|More],Y,[X|Z]):-
    member(X,Y),
    intersection(More,Y,Z).
intersection([X|More],Y,Z):-
    intersection(More,Y,Z).
```

Si rappresenti l'albero SLD relativo al goal

```
:- intersection([1,2],[2,3],L).
```

e si indichino i rami di successo. Si indichi come l'utilizzo del cut (!) possa portare alla definizione corretta del predicato intersezione.



Con l'utilizzo del cut, si elimina il ramo che porta al successo con lista L=[] (soluzione sbagliata) riportata in

figura nel quadrato. Quindi intersection/3 con il cut si comporta correttamente.

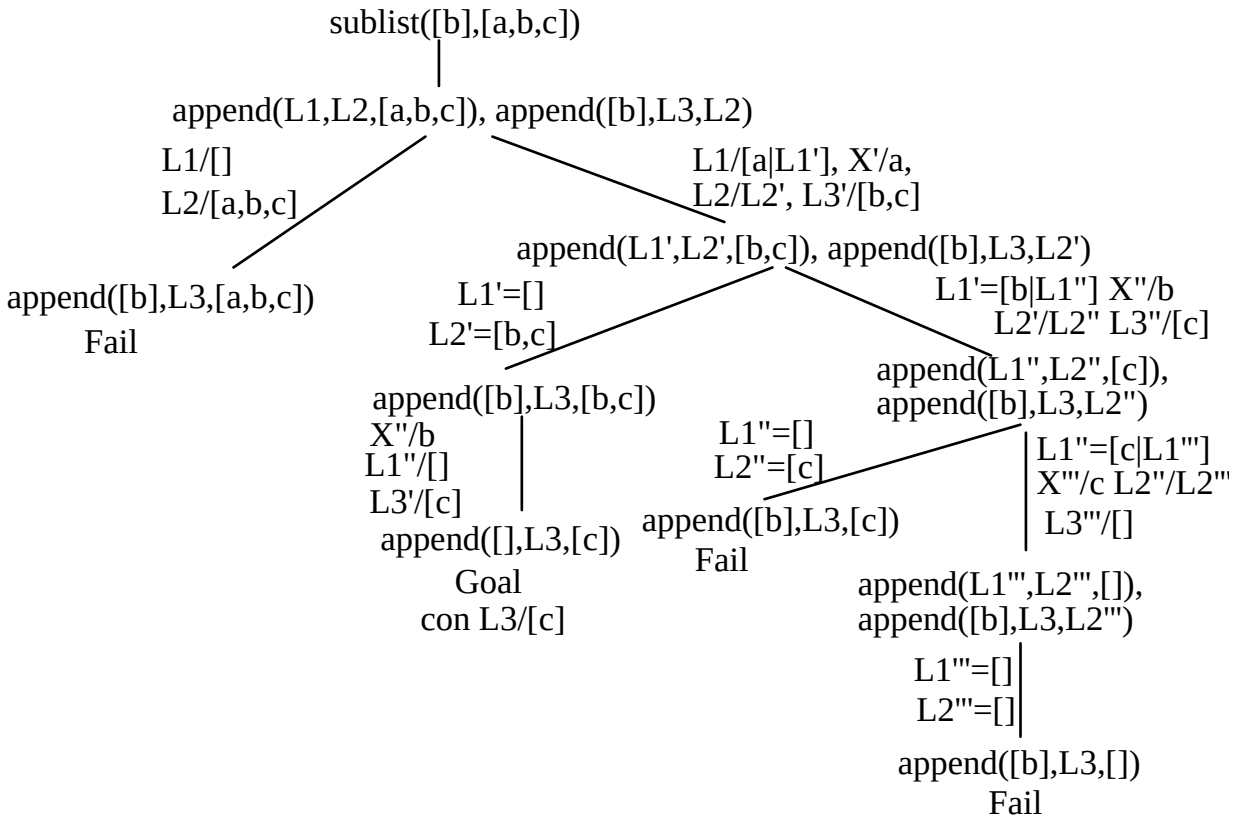
Esercizio

Dato il predicato `append` ed il seguente programma Prolog:

```
sublist(S,L) :-
    append(L1,L2,L),
    append(S,L3,L2).
```

Si mostri l'albero SLD con derivazione left-most relativo al goal: `sublist([b], [a,b,c]).`

```
append([],L,L).
append([X|L1],L2,[X|L3]):- append(L1,L2,L3).
```



Esercizio

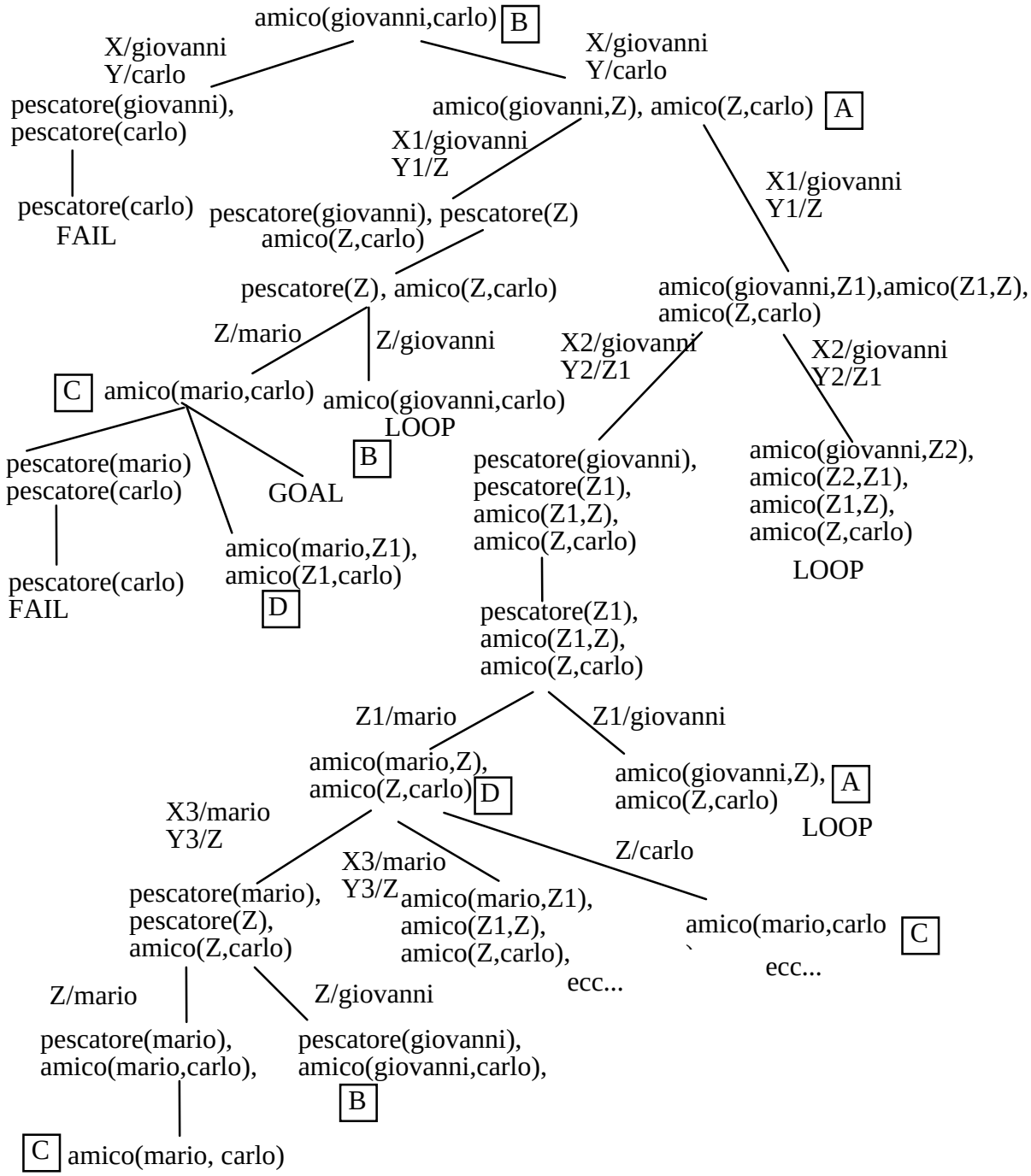
Si consideri il seguente programma Prolog:

```
amico(X,Y):- pescatore(X), pescatore(Y).
amico(X,Y):- amico(X,Z),amico(Z,Y).
```

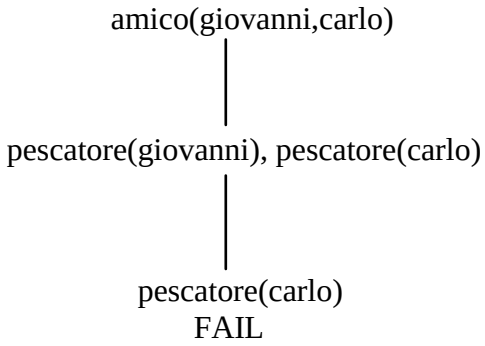
Si rappresenti l'albero SLD relativo al goal
`:- amico(giovanni, carlo)`
assumendo la presenza, all'inizio del database, dei fatti:

```
pescatore(mario).
pescatore(giovanni).
amico(mario,carlo).
```

Si supponga di introdurre un cut all'inizio della prima clausola e si discuta l'effetto del cut sull'albero SLD


amico(giovanni,carlo) B
X/giovanni
Y/carlo
X/giovanni
Y/carlo
pescatore(giovanni),
pescatore(carlo)
amico(giovanni,Z), amico(Z,carlo) A
X1/giovanni
Y1/Z
pescatore(carlo)
FAIL
pescatore(giovanni), pescatore(Z)
amico(Z,carlo)
X1/giovanni
Y1/Z
pescatore(Z), amico(Z,carlo)
amico(giovanni,Z1),amico(Z1,Z),
amico(Z,carlo)
Z/mario
Z/giovanni
X2/giovanni
Y2/Z1
X2/giovanni
Y2/Z1
C amico(mario,carlo)
amico(giovanni,carlo)
LOOP
B
pescatore(mario)
pescatore(carlo)
GOAL
pescatore(giovanni),
pescatore(Z1),
amico(Z1,Z),
amico(Z,carlo)
amico(giovanni,Z2),
amico(Z2,Z1),
amico(Z1,Z),
amico(Z,carlo)
LOOP
amico(mario,Z1),
amico(Z1,carlo)
D
pescatore(carlo)
FAIL
pescatore(Z1),
amico(Z1,Z),
amico(Z,carlo)
Z1/mario
Z1/giovanni
amico(mario,Z),
amico(Z,carlo) D
amico(giovanni,Z),
amico(Z,carlo)
A
LOOP
X3/mario
Y3/Z
Z/carlo
X3/mario
Y3/Z
pescatore(mario),
pescatore(Z),
amico(Z,carlo)
amico(mario,Z1),
amico(Z1,Z),
amico(Z,carlo),
ecc...
amico(mario,carlo
`
ecc...
C
Z/mario
Z/giovanni
pescatore(mario),
amico(mario,carlo),
pescatore(giovanni),
amico(giovanni,carlo),
B
C amico(mario, carlo)

L'effetto del cut all'inizio della prima clausola è il seguente: il programma esegue la prima clausola riportando fallimento. Infatti, mentre pescatore(giovanni) ha successo, pescatore(carlo) fallisce. Il cut ha l'effetto di eliminare il choice point di amico/2 e quindi il programma non trova nemmeno una soluzione.
L'albero SLD è il seguente:



Esercizio

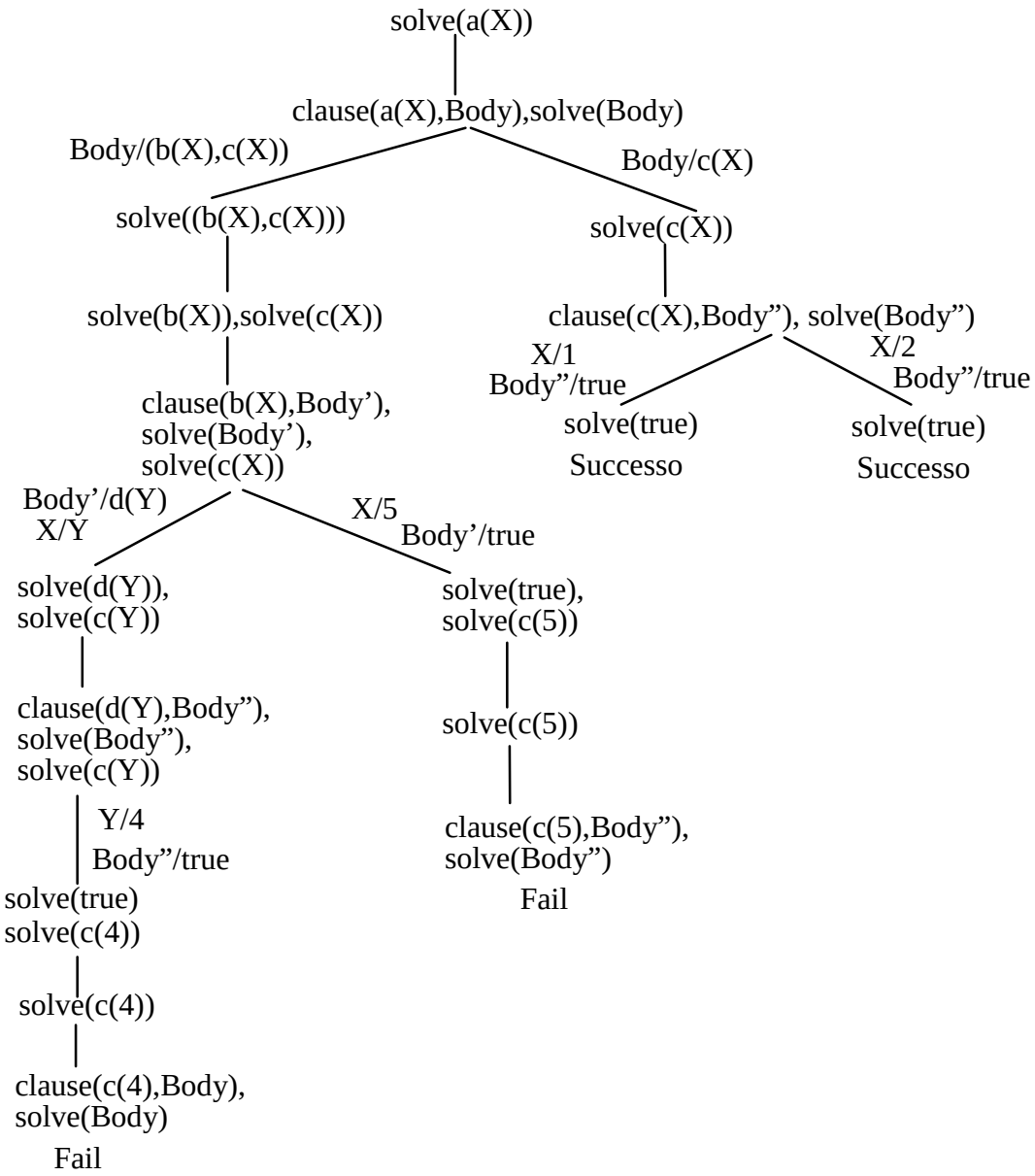
Dato il seguente metainterprete Prolog:

```
solve(true):-!.
solve((A,B)):- !,solve(A), solve(B).
solve(A):- clause(A,Body), solve(Body).
```

e il programma Prolog:

```
a(X):- b(X),c(X).
a(X):- c(X).
b(Y):- d(Y).
b(5).
c(1).
c(2).
d(4).
```

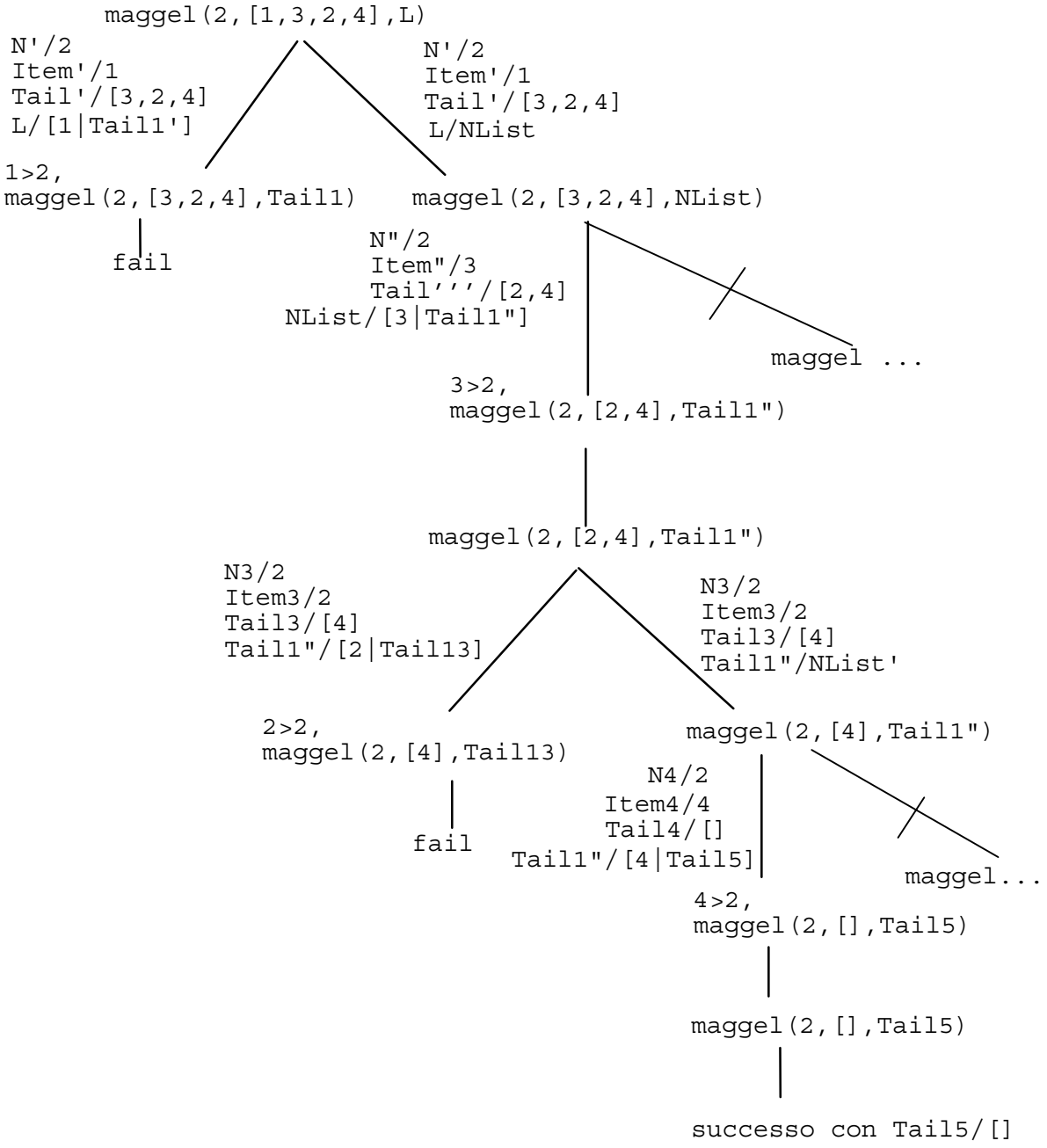
Si mostri l'albero SLD con derivazione left-most relativo al goal: `solve(a(X)).`


solve(a(X))
clause(a(X),Body),solve(Body)
Body/(b(X),c(X))
Body/c(X)
solve((b(X),c(X)))
solve(c(X))
solve(b(X)),solve(c(X))
clause(c(X),Body”), solve(Body”)
X/1
Body”/true
X/2
Body”/true
clause(b(X),Body’),
solve(Body’),
solve(c(X))
solve(true)
Successo
solve(true)
Successo
Body’/d(Y)
X/Y
X/5
Body’/true
solve(d(Y)),
solve(c(Y))
solve(true),
solve(c(5))
clause(d(Y),Body”),
solve(Body”),
solve(c(Y))
solve(c(5))
Y/4
Body”/true
clause(c(5),Body”),
solve(Body”)
Fail
solve(true)
solve(c(4))
solve(c(4))
clause(c(4),Body),
solve(Body)
Fail

Esercizio

Si definisca il predicato `maggel(N, List, Nlist)`, che dato un numero `N` ed una lista `List` restituisca in `Nlist` una lista composta dai numeri presenti in `List` maggiori di `N`. Si mostri poi l'albero SLD relativo alla query:
`:-maggel(2, [1,3,2,4],L).`
utilizzando la regola di selezione left-most. Se nel codice si utilizza un cut, lo si consideri nella costruzione dell'albero.

```
maggel(N,[],[]).
maggel(N,[Item|Tail],[Item|Tail1]):-
    Item > N,
    !,maggel(N,Tail,Tail1).
maggel(N,[Item|Tail],NList):-
    maggel(N,Tail,NList).
```

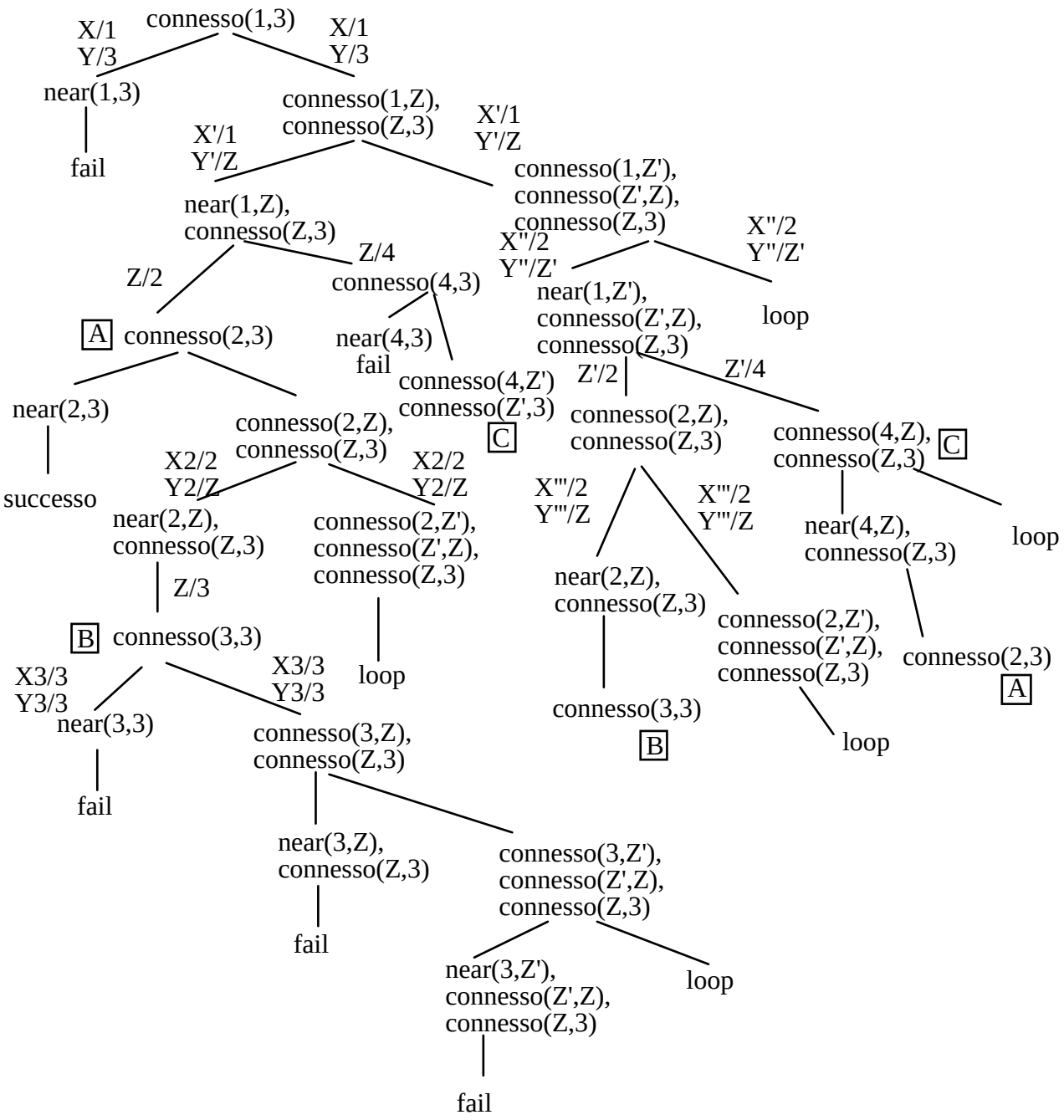

maggel(2,[1,3,2,4],L)
N'/2
Item'/1
Tail'/[3,2,4]
L/[1|Tail1']
N'/2
Item'/1
Tail'/[3,2,4]
L/NList
1>2,
maggel(2,[3,2,4],Tail1)
maggel(2,[3,2,4],NList)
fail
N"/2
Item"/3
Tail'''/[2,4]
NList/[3|Tail1"]
maggel ...
3>2,
maggel(2,[2,4],Tail1")
maggel(2,[2,4],Tail1")
N3/2
Item3/2
Tail3/[4]
Tail1"/[2|Tail13]
N3/2
Item3/2
Tail3/[4]
Tail1"/NList'
2>2,
maggel(2,[4],Tail13)
maggel(2,[4],Tail1")
fail
N4/2
Item4/4
Tail4/[]
Tail1"/[4|Tail5]
maggel...
4>2,
maggel(2,[],Tail5)
maggel(2,[],Tail5)
successo con Tail5/[]

Esercizio

Si considerino le seguenti regole:

```
connesso(X,Y):- near(X,Y).
connesso(X,Y):- connesso(X,Z), connesso(Z,Y).
near(1,2).
near(2,3).
near(1,4).
near(4,2).
```

Si mostri l'albero SLD relativo alla query: `connesso(1,3)` utilizzando la regola di selezione **left-most** .

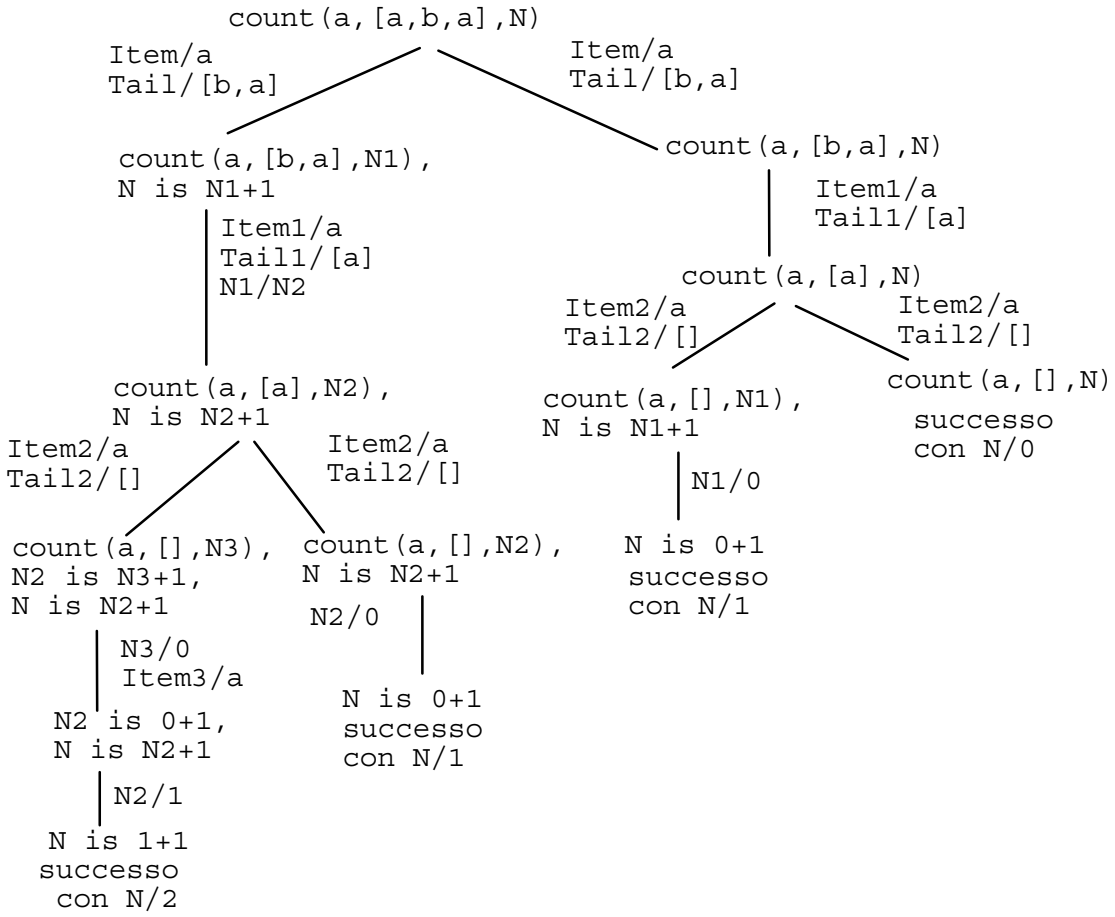

connesso(1,3)
X/1
Y/3
X/1
Y/3
near(1,3)
fail
connesso(1,Z),
connesso(Z,3)
X'/1
Y'/Z
X'/1
Y'/Z
near(1,Z),
connesso(Z,3)
connesso(1,Z'),
connesso(Z',Z),
connesso(Z,3)
Z/2
Z/4
connesso(4,3)
X''/2
Y''/Z'
X''/2
Y''/Z'
A
connesso(2,3)
near(4,3)
fail
near(1,Z'),
connesso(Z',Z),
connesso(Z,3)
loop
connesso(4,Z')
connesso(Z',3)
C
Z'/2
Z'/4
near(2,3)
connesso(2,Z),
connesso(Z,3)
connesso(2,Z),
connesso(Z,3)
connesso(4,Z),
connesso(Z,3)
C
X2/2
Y2/Z
X2/2
Y2/Z
X'''/2
Y'''/Z
X'''/2
Y'''/Z
successo
near(2,Z),
connesso(Z,3)
connesso(2,Z'),
connesso(Z',Z),
connesso(Z,3)
near(4,Z),
connesso(Z,3)
loop
Z/3
near(2,Z),
connesso(Z,3)
connesso(2,Z'),
connesso(Z',Z),
connesso(Z,3)
B
connesso(3,3)
X3/3
Y3/3
X3/3
Y3/3
loop
connesso(2,3)
A
near(3,3)
connesso(3,3)
B
loop
fail
connesso(3,Z),
connesso(Z,3)
near(3,Z),
connesso(Z,3)
connesso(3,Z'),
connesso(Z',Z),
connesso(Z,3)
fail
near(3,Z'),
connesso(Z',Z),
connesso(Z,3)
loop
fail

Esercizio

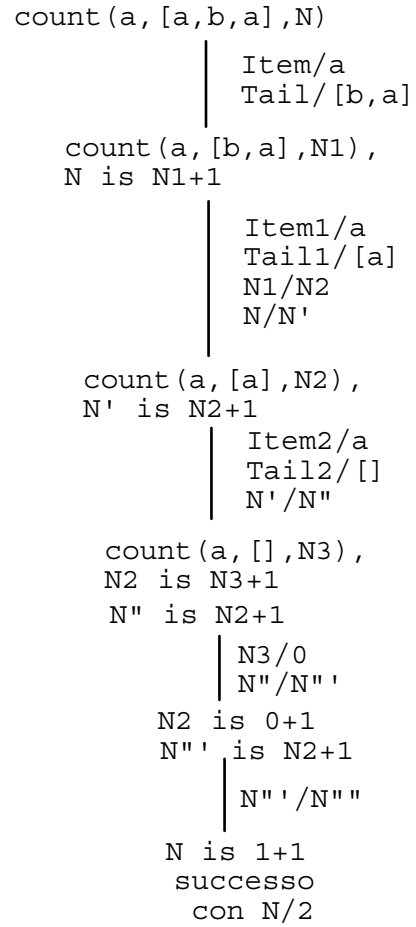
Si definisca il predicato `count(Item, List, N)`, che dato un item `Item` ed una lista `List` restituisca il numero `N` di volte in cui `Item` compare nella lista.
Si mostri poi l'albero SLD relativo alla query: `count(a,[a,b,a],N)` utilizzando la regola di selezione left-most . Se nel codice si utilizza il cut, si mostrino le versioni dell'albero SLD con e senza il cut.

```
count(Item,[],0).
count(Item,[Item|Tail],N):-
    !,count(Item,Tail,N1), N is N1 +1.
count(Item,[_|Tail],N):- count(Item,Tail,N).
```

Versione senza cut:



Versione con cut:

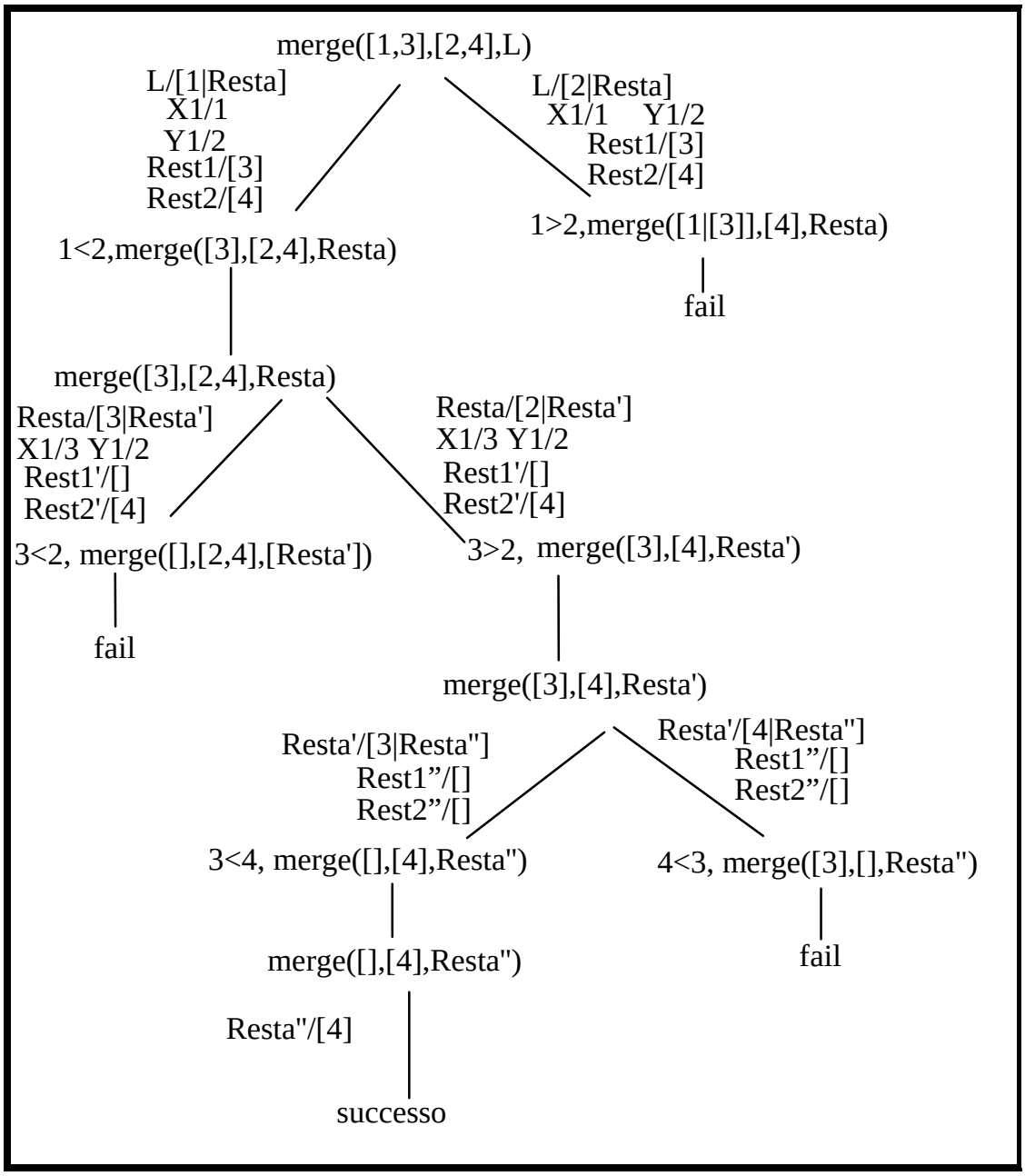


Esercizio

Si consideri il seguente programma Prolog:

```
merge([],L2,L2).
merge(L1,[],L1).
merge([X|Rest1],[Y|Rest2],[X|Rest]):-
   X<Y,merge(Rest1,[Y|Rest2],Rest).
merge([X|Rest1],[Y|Rest2],[Y|Rest]):-
   X>Y,merge([X|Rest1],Rest2,Rest).
```

Si mostri l'albero SLD con derivazione left-most relativo al goal: `merge([1,3],[2,4],L).`

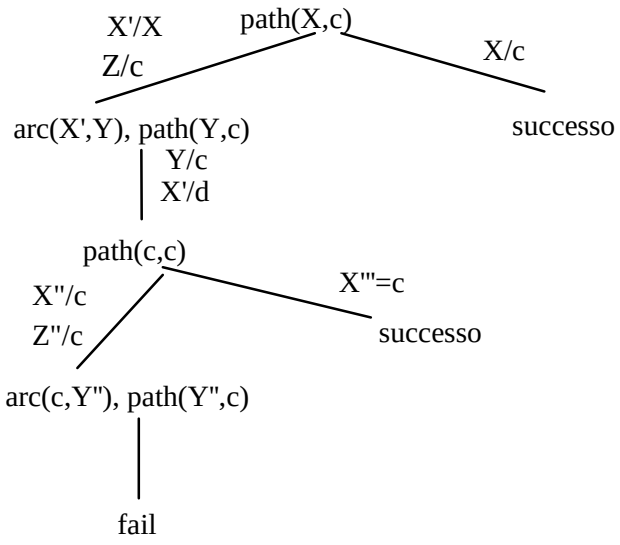

merge([1,3],[2,4],L)
L/[1|Resta]
X1/1
Y1/2
Rest1/[3]
Rest2/[4]
L/[2|Resta]
X1/1 Y1/2
Rest1/[3]
Rest2/[4]
1<2,merge([3],[2,4],Resta)
1>2,merge([1|[3]],[4],Resta)
fail
merge([3],[2,4],Resta)
Resta/[3|Resta']
X1/3 Y1/2
Rest1'/[]
Rest2'/[4]
Resta/[2|Resta']
X1/3 Y1/2
Rest1'/[]
Rest2'/[4]
3<2, merge([],[2,4],[Resta'])
3>2, merge([3],[4],Resta')
fail
merge([3],[4],Resta')
Resta'/[3|Resta"]
Rest1”/[]
Rest2”/[]
Resta'/[4|Resta"]
Rest1”/[]
Rest2”/[]
3<4, merge([],[4],Resta")
4<3, merge([3],[],Resta")
merge([],[4],Resta")
fail
Resta"/[4]
successo

Esercizio

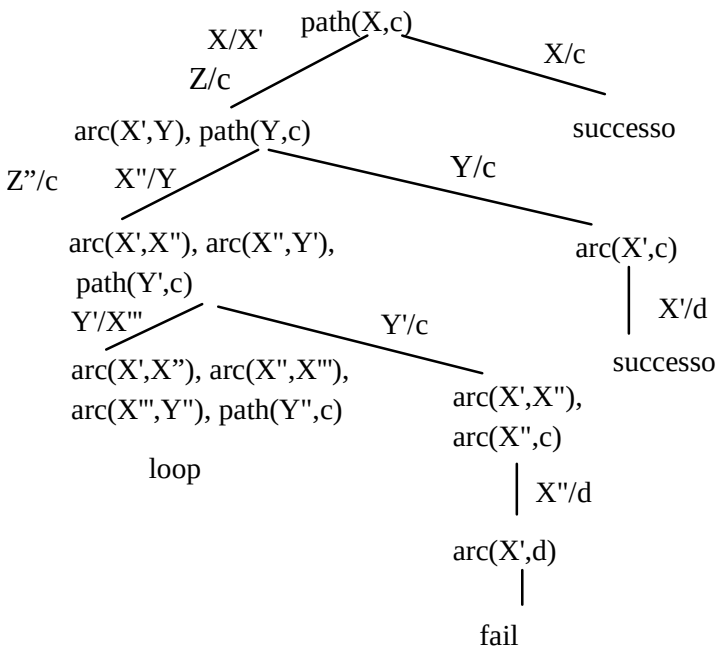
Si consideri il seguente programma Prolog:

```
path(X,Z):-arc(X,Y),path(Y,Z).
path(X,X).
arc(d,c).
```

Si mostrino i due alberi SLD relativi alla query: `path(X,c)` che utilizzano, rispettivamente, la regola di selezione right-most e left-most e si commentino i risultati in base al teorema che riguarda l'indipendenza della regola di selezione.
Albero sld: strategia di selezione left-most:



Albero sld: strategia di selezione right-most:



Nei due casi la struttura dell'albero (i loop e i rami di fallimento) cambia. Rimangono inalterati i rami di successo che portano alle medesime soluzioni.

Esercizio

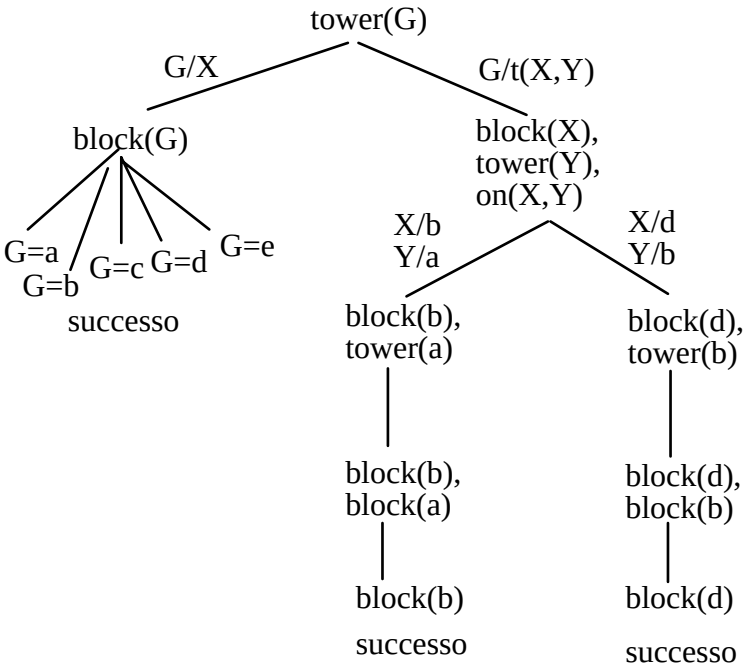
Si consideri il seguente programma Prolog:

```
tower(X):-block(X).
tower(t(X,Y)):-block(X),tower(Y),on(X,Y).

block(a).
block(b).
block(c).
block(d).
block(e).

on(b,a).
on(d,b).
```

Si mostri l'albero SLD relativo relativo alla regola di selezione **right-most** per la query: `tower(G)`



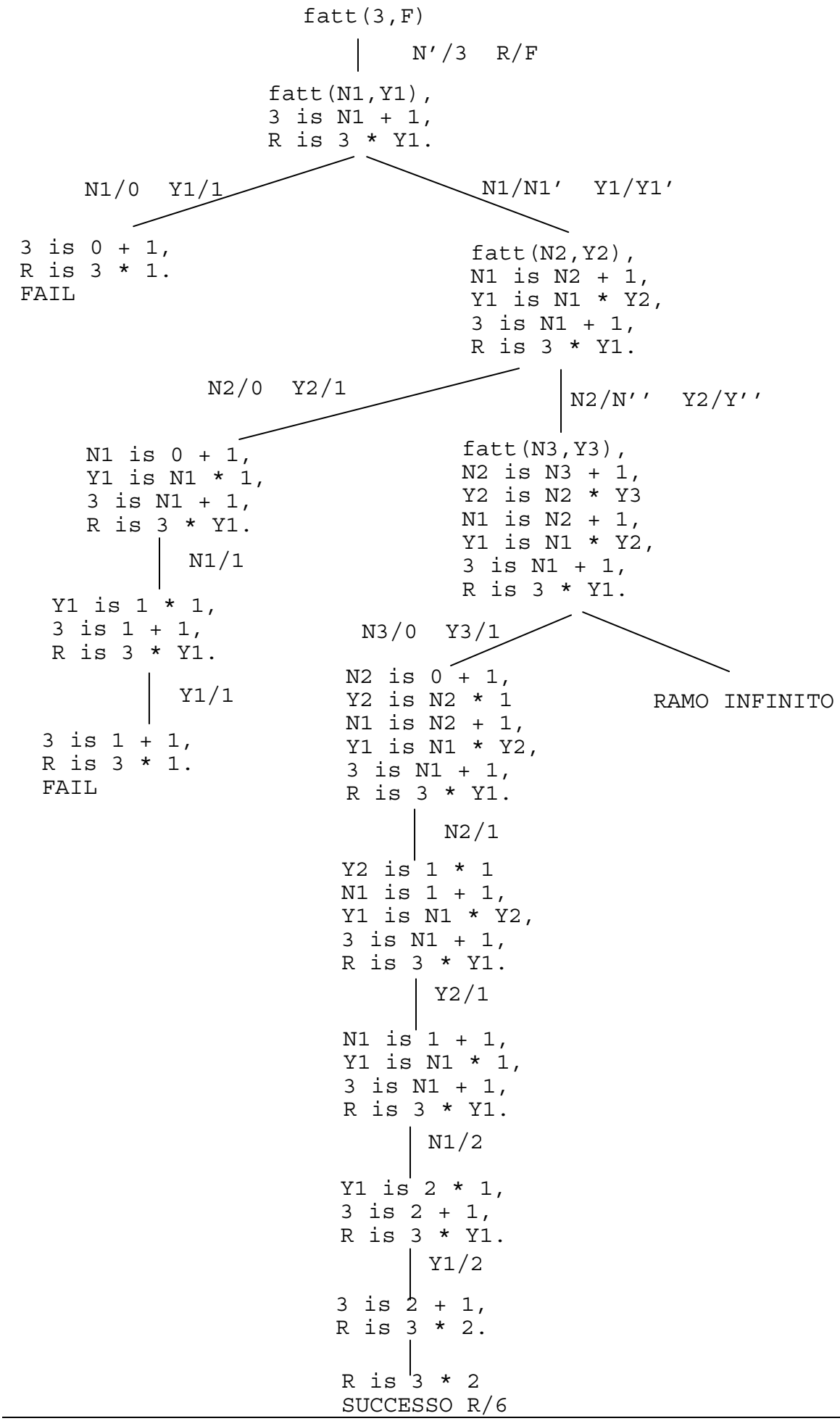
Esercizio

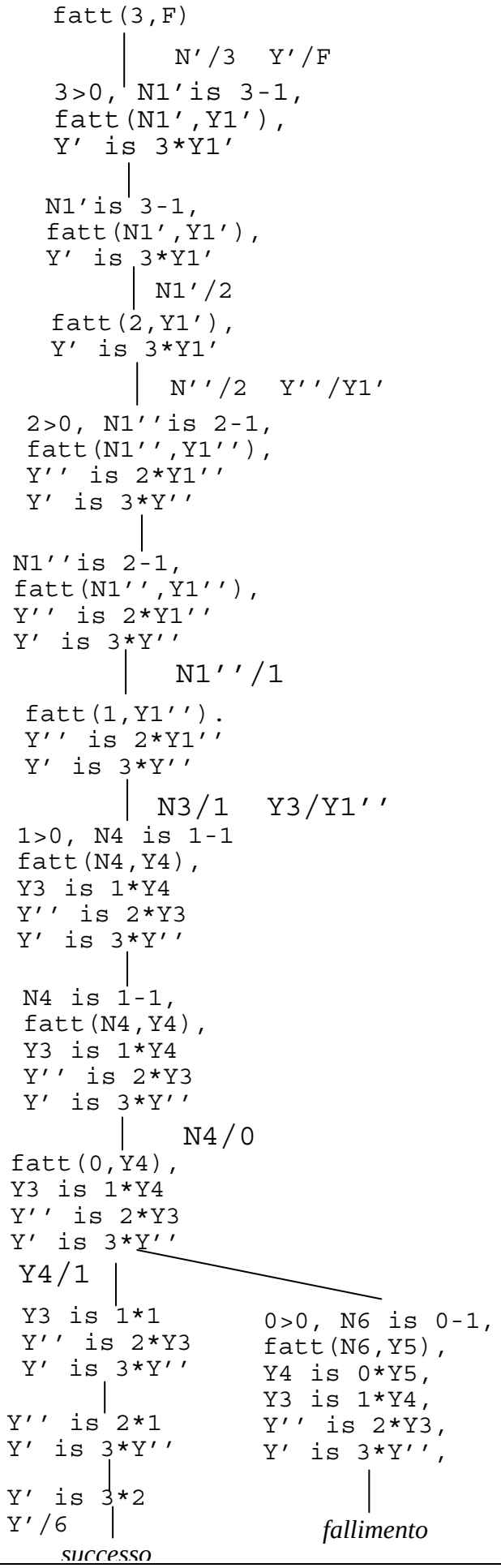
Si considerino le due versioni seguenti per il calcolo del fattoriale

```
fatt(0,1).
fatt(N,Y):-
        fatt(N1,Y1),
        N is N1 + 1,
        Y is N * Y1.
```

```
fatt(0,1).
fatt(N,Y):- N>0,
        N1 is N - 1,
        fatt(N1,Y1),
        Y is N * Y1.
```

Si mostri l'albero SLD in entrambi i casi relativo alla regola di selezione **left-most** per la query: `fatt(3,F)`


fatt(3,F)
N'/3 R/F
fatt(N1,Y1),
3 is N1 + 1,
R is 3 * Y1.
N1/0 Y1/1
N1/N1' Y1/Y1'
3 is 0 + 1,
R is 3 * 1.
FAIL
fatt(N2,Y2),
N1 is N2 + 1,
Y1 is N1 * Y2,
3 is N1 + 1,
R is 3 * Y1.
N2/0 Y2/1
N2/N'' Y2/Y''
N1 is 0 + 1,
Y1 is N1 * 1,
3 is N1 + 1,
R is 3 * Y1.
N1/1
fatt(N3,Y3),
N2 is N3 + 1,
Y2 is N2 * Y3
N1 is N2 + 1,
Y1 is N1 * Y2,
3 is N1 + 1,
R is 3 * Y1.
Y1 is 1 * 1,
3 is 1 + 1,
R is 3 * Y1.
Y1/1
N3/0 Y3/1
3 is 1 + 1,
R is 3 * 1.
FAIL
N2 is 0 + 1,
Y2 is N2 * 1
N1 is N2 + 1,
Y1 is N1 * Y2,
3 is N1 + 1,
R is 3 * Y1.
RAMO INFINITO
N2/1
Y2 is 1 * 1
N1 is 1 + 1,
Y1 is N1 * Y2,
3 is N1 + 1,
R is 3 * Y1.
Y2/1
N1 is 1 + 1,
Y1 is N1 * 1,
3 is N1 + 1,
R is 3 * Y1.
N1/2
Y1 is 2 * 1,
3 is 2 + 1,
R is 3 * Y1.
Y1/2
3 is 2 + 1,
R is 3 * 2.
R is 3 * 2
SUCCESSO R/6


fatt(3,F)
N'/3 Y'/F
3>0, N1'is 3-1,
fatt(N1',Y1'),
Y' is 3*Y1'
N1'is 3-1,
fatt(N1',Y1'),
Y' is 3*Y1'
N1'/2
fatt(2,Y1'),
Y' is 3*Y1'
N''/2 Y''/Y1'
2>0, N1''is 2-1,
fatt(N1'',Y1''),
Y'' is 2*Y1''
Y' is 3*Y''
N1''is 2-1,
fatt(N1'',Y1''),
Y'' is 2*Y1''
Y' is 3*Y''
N1''/1
fatt(1,Y1'').
Y'' is 2*Y1''
Y' is 3*Y''
N3/1 Y3/Y1''
1>0, N4 is 1-1
fatt(N4,Y4),
Y3 is 1*Y4
Y'' is 2*Y3
Y' is 3*Y''
N4 is 1-1,
fatt(N4,Y4),
Y3 is 1*Y4
Y'' is 2*Y3
Y' is 3*Y''
N4/0
fatt(0,Y4),
Y3 is 1*Y4
Y'' is 2*Y3
Y' is 3*Y''
Y4/1
Y3 is 1*1
Y'' is 2*Y3
Y' is 3*Y''
Y'' is 2*1
Y' is 3*Y''
Y' is 3*2
Y'/6
successo
0>0, N6 is 0-1,
fatt(N6,Y5),
Y4 is 0*Y5,
Y3 is 1*Y4,
Y'' is 2*Y3,
Y' is 3*Y'',
fallimento