UNIVERSITY OF PATRAS
DEPT. OF COMPUTER ENGINEERING & INFORMATICS
ARTIFICIAL INTELLIGENCE

SOLUTIONS of 4th Assignment

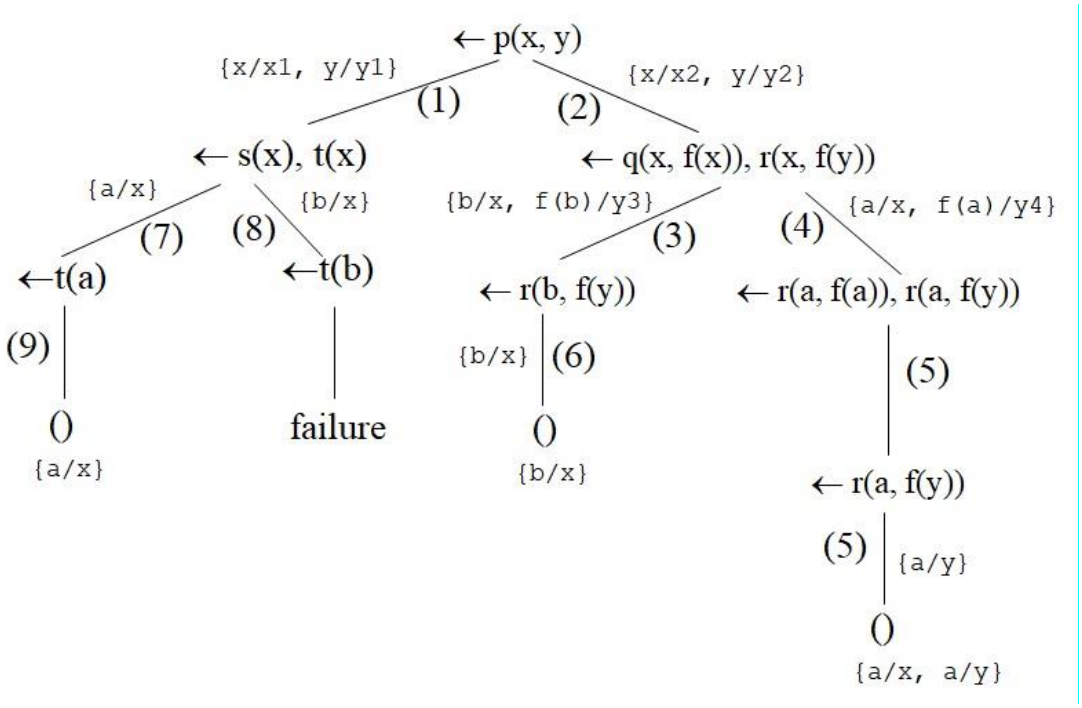
1. The following is a definite program:

(1) p(x1, y1) ← s(x1), t(x1)}
(2) p(x2, y2) ← q(x2, f(x2)), r(x2, f(y2))
(3) q(b, y3)
(4) q(a, y4) ← r(a, f(a))
(5) r(a, f(a))
(6) r(b, f(b))
(7) s(a)
(8) s(b)
(9) t(a).

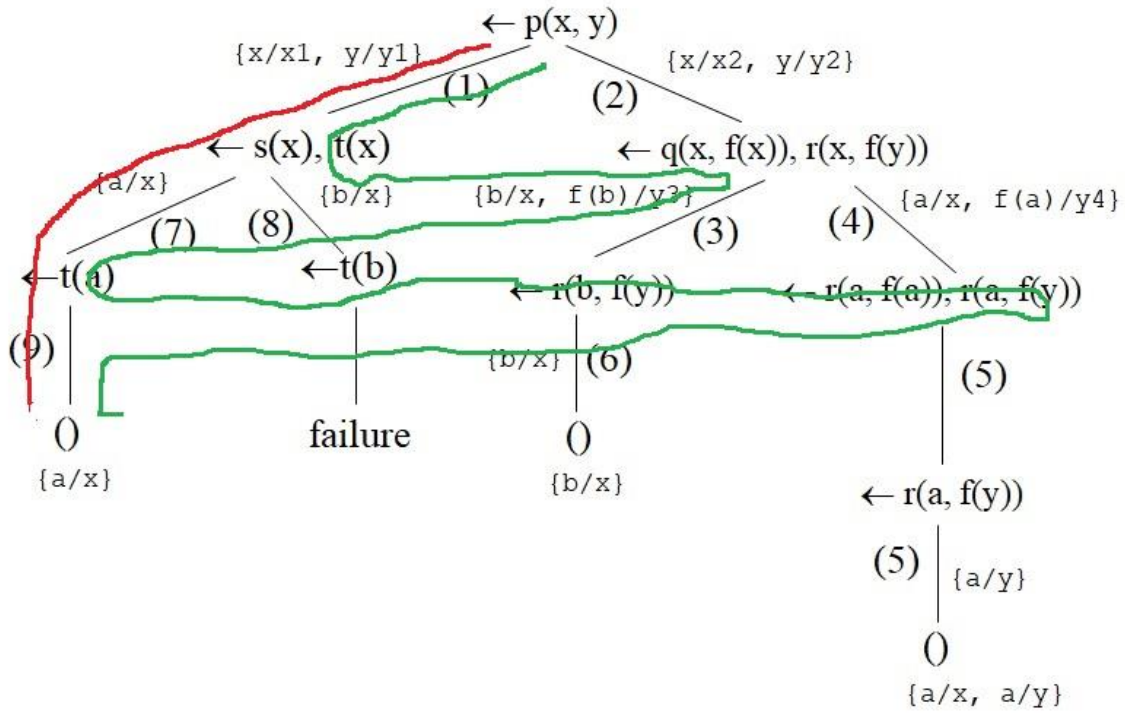
(a) Given the goal "← p(x, y)", sketch its SLD-tree using Prolog's computation rule.
(b) Which one from "depth-first with backtracking" and "breadth-first" strategies reaches the solution faster?
(c) What changes and where we should make to reach the solution faster in both cases?

Answers

(a) Given the goal "← p(x, y)", sketch its SLD-tree using Prolog's computation rule.



(b) Which one from "depth-first with backtracking" and "breadth-first" strategies reaches the solution faster?



In red is the route followed by DFS (depth-first search), whereas in green is the route followed by BFS (breadth-first search). So, DFS finds a solution much faster (3 steps, in this case) than BFS (7 steps).

(c) What changes and where we should make to reach the solution faster in both cases?

We can do nothing to make them reach a solution faster. This is the best scenario. Usually, we change the order of formulas in the program, to change the place of solution paths in the SLD tree. However, in this case there is no such change that can help.

2. The following set of formulas in Clause Normal Form is given (we omit disjunction symbols):

{R, ¬Q, ¬T}, {R, ¬Q, ¬P}, {¬S, T, P}, {Q}, {S}, {¬T}

Use resolution refutation to prove that "R" is a theorem.

Answers

To prove R is theorem, we negate it, convert it to CNF, put it in the set of formulas and try to produce the empty clause using resolution.

So, R → ¬R → {¬R} and our final set of formulas is:

(1) {R, ¬Q, ¬T}
(2) {R, ¬Q, ¬P}
(3) {¬S, T, P}
(4) {Q}
(5) {S}
(6) {¬T}
(7) {¬R}

We now make resolutions between the formulas until we produce the empty clause.

(8) **{¬Q, ¬T}**, produced by resolving (1) and (7)
(9) **{¬Q, ¬P}**, produced by resolving (2) and (7)
(10) **{¬T}**, produced by resolving (4) and (8)
(11) **{¬P**}, produced by resolving (4) and (9)
(12) **{¬S, P}**, produced by resolving (10) and (3)
(13) **{¬S}**, produced by resolving (11) and (12)
(14) **{}**, produced by resolving (13) and (5)

So, R is a theorem.

3. The following facts are given: "Paul is father of John and Georgia" and "Helen is mother of Maria and Peter". Also, we are given the following rule type knowledge: "Two humans are siblings if they have the same mother or the same father".

(a) Represent the above knowledge as Prolog statements-program.
(b) What will be the answer to the question: "Who are the siblings of John?" (implement the question in Prolog and show the inference flow).
(c) If the answer in (b) includes "John" himself, then modify your program to remove this case.

Answers

(a) Represent the above knowledge as Prolog statements-program.

```
is_father(paul, john).
is_father(paul, georgia).
is_mother(helen, maria).
is_mother(helen, peter).
is_sibling(X, Y) :- is_father(Z, X), is_father(Z, Y).
is_sibling(X, Y) :- is_mother(Z, X), is_mother(Z, Y).
```

(b) What will be the answer to the question: "Who are the siblings of John?" (implement the question in Prolog and show the inference flow).

We set the question in Prolog:

```
?- is_sibling(X, john). /* "?-" is the Prolog prompt) */
```

then the system will respond with

```
X = john
```

If we type an ";" (which means continue to find another solution)

```
X = john;          /* notice that this means john is sibling of himself */
```

the system will respond

```
X = georgia
```

And if we ask for a further solution:

```
X = Georgia;
```

It will respond:

```
No
```

(which here means no other solutions).

(c) If the answer in (b) includes "John" himself, then modify your program to remove this case.

The modification concerns the rules of the program:

```
is_sibling(X, Y) :- is_father(Z, X), is_father(Z, Y), not(X=Y).
is_sibling(X, Y) :- is_mother(Z, X), is_mother(Z, Y) , not(X=Y).
```

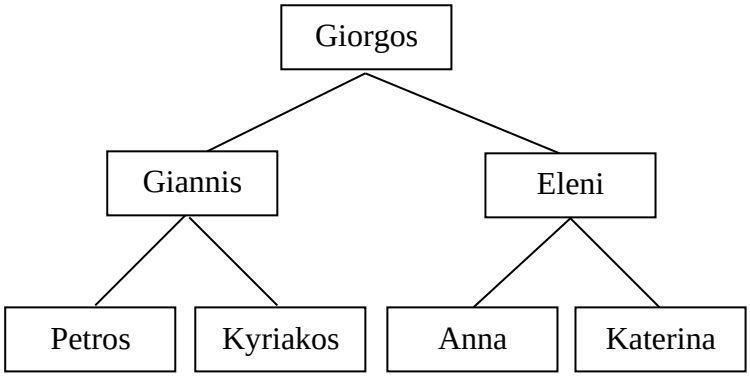
we demand that X cannot be equal to Y, which solves the problem.

4. The following binary tree is given for which you are asked to

(a) Write a Prolog program that represents that tree as a series of facts, using the predicate "tree/3".

(b) Add appropriate rules to the program so that it passes the tree in a depth-first manner printing out the names of the nodes (i.e. Giorgos-Giannis-Petros-Kyriakos-Eleni-Anna-Katerina). The program should start using the predicate go/0.

(c) The same as (b) for breadth-first pass (i.e. Giorgos-Giannis-Eleni-Petros-Kyriakos-Anna-Katerina).



Answers

(a) Write a Prolog program that represents that tree as a series of facts, using the predicate "tree/3".

```
tree(giorgos, giannis, eleni).
tree(giannis, petros, kyriakos).
tree(eleni, anna, katerina).
tree(petros,nil,nil).
tree(kyriakos,nil,nil).
tree(anna,nil,nil).
tree(katerina,nil,nil).
```

(b) Add appropriate rules to the program so that it passes the tree in a depth-first manner printing out the names of the nodes (i.e. Giorgos-Giannis-Petros-Kyriakos-Eleni-Anna-Katerina). The program should start using the predicate go/0.

```
print_tree(X):-
tree(X,Y,Z),
write(X), nl,
print_tree(Y),
print_tree(Z).
print_tree(X):-tree(X,nil,nil).
go:- write('Give Root: '), read(T), nl, print_tree(T)
```

(c) The same as (b) for breadth-first pass (i.e. Giorgos-Giannis-Eleni-Petros-Kyriakos-Anna-Katerina).

```
print_br(X):- write(X), nl, print_tree(X).
```

```
print_tree(X):- tree(X,nil,nil).
print_tree(X):- tree(X,Y,Z),
write(Y), nl,
write(Z), nl,
print_tree(Y),
print_tree(Z).
go:- write('Give Root: '), read(T), nl, print_br(T).
```