CSE 130: Programming Languages – Principles and Paradigms Fall 2000

Problem Set 3 - Solutions

Instructor: Daniele Micciancio Nov. 15, 2000

Problem 1 (5 points)

- substitution: $X = f(a,a), Y = a, Z = a, W = a$; instance: $g(f(a,a), f(a,a))$
- the terms do not unify;
- substitution $X = a, Y = Z, V = f(a,Z), W = a$; instance: $f(g(a,a,Z), f(a,Z))$

Problem 2 (10 points)

The trees corresponding to the fourth programs in this problems are depicted in the pictures that follow. For simplicity we wrote "c" instead of "connected". On the arrows, the rule that is applied together with the unifying substitution is indicated.

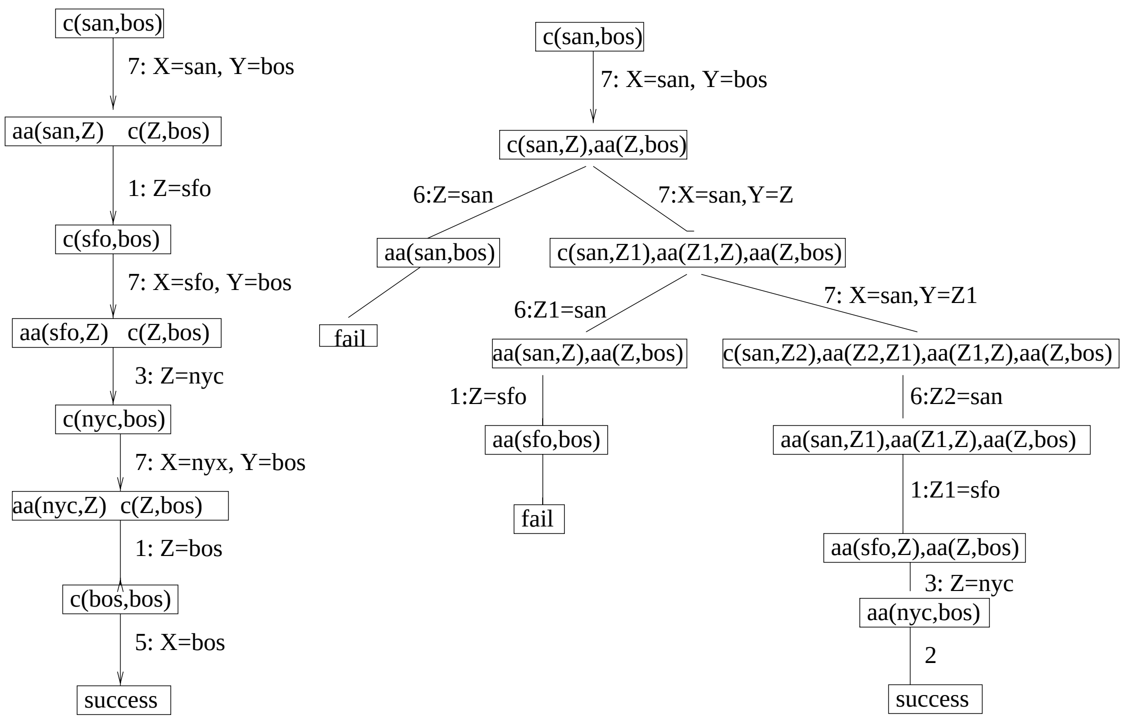

Figure 1: Execution tree for programs 1 and 2

The following figure contains the execution trees for programs 3 and 4. One can observe that program 3 loops. The reason is that in order to solve the goal `c(san,Zi)`, rule 6 is applied and this leads to a similar goal: `c(san, Z(i+1))`.

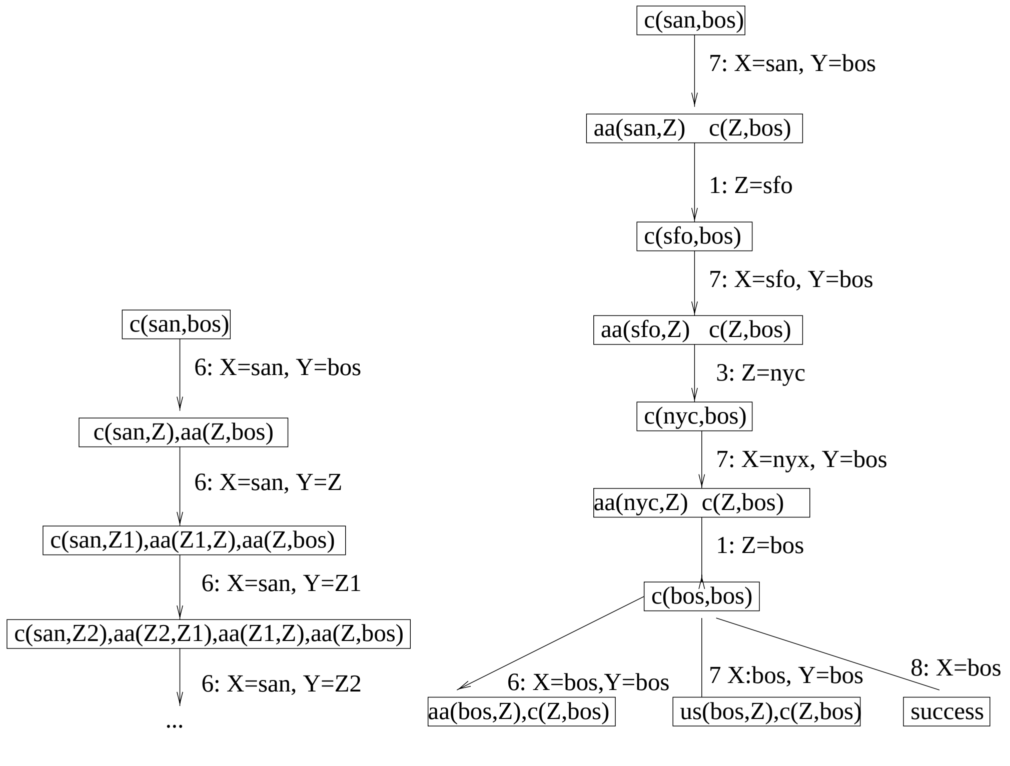

Figure 2: Execution tree for programs 1 and 2

Problem 3 (15 points)

(a) The predicate `remove` is defined as follows:

```
remove(X,[X|L],L).
remove(X,[Y|L],[Y|L1]):-remove(X,L,L1).
```

When called on (X,L1,L2), the predicate holds true if X is in the list L1 and L2 is L1 in which the first occurence of X has been removed. Now, one can define permute as follows:

```
permuted([],[]).
permuted(X,[Y|L]):-remove(Y,X,X1),permuted(X1,L).
```

(b) The predicate *insert*, on input a number X and a ordered list Ys should return in Zs the result of inserting X in Ys. Inserting X should preserve the order.

One can define *insert* as follows:

```
insert(X,[],[X]).
insert(X,[Y|Ys],[X|[Y|Ys]]):-X<Y.
insert(X,[Y|Ys],[Y|Zs]):-Y=<X,insert(X,Ys,Zs).
```

(c) The predicate*shuffle*(As,Bs,Xs), should split the list Xs in two smaller lists of aproximately equal size. One possible definition is the following (the list is split in the list of its odd order elements and the list of its even order elements).

```
shuffle([],[],[]).
shuffle([X],[],[X]).
shuffle([X|L1],[Y|L2],[X,Y|L]):-shuffle(L1,L2,L).
```

The predicate *merge* should take two lists (presumably sorted), Xs and Ys, and produce Zs which has the same elements as the union of elements of Xs and Ys and which are sorted.

```
merge([],L,L).
merge(L,[],L).
merge([X|Xs],[Y|Ys],[X|Rs]):-X=<Y,merge(Xs,[Y|Ys],Rs).
merge([X|Xs],[Y|Ys],[Y|Rs]):-Y=<X,merge([X|Xs],Ys,Rs).
```