

Introduction to the Theory of Computation

Jean Gallier

Homework 5b

March 25, 2004; Due April 13 2004

Programming Project (80 pts). This problem is based on the method proved correct in Problem B2 of Homework 4.

Given a DFA $D = (Q, \Sigma, \delta, q_0, F)$, for any two states $p, q \in Q$, a fast algorithm for computing the forward closure of the relation $R = \{(p, q)\}$, or detecting a bad pair of states, can be obtained as follows: An equivalence relation on Q is represented by a partition Π . Each equivalence class C in the partition is represented by a tree structure consisting of nodes and (parent) pointers, with the pointers from the sons of a node to the node itself. The root has a null pointer. Each node also maintains a counter keeping track of the number of nodes in the subtree rooted at that node.

Two functions *union* and *find* are defined as follows. Given a state p , $find(p, \Pi)$ finds the root of the tree containing p as a node (not necessarily a leaf). Given two root nodes p, q , $union(p, q, \Pi)$ forms a new partition by merging the two trees with roots p and q as follows: if the counter of p is smaller than that of q , then let the root of p point to q , else let the root of q point to p .

In order to speed up the algorithm, we can modify *find* as follows: during a call $find(p, \Pi)$, as we follow the path from p to the root r of the tree containing p , we redirect the parent pointer of every node q on the path from p (including p itself) to r .

Say that a pair $\langle p, q \rangle$ is *bad* iff either both $p \in F$ and $q \notin F$, or both $p \notin F$ and $q \in F$. The function *bad* is such that $bad(\langle p, q \rangle) = true$ if $\langle p, q \rangle$ is bad, and $bad(\langle p, q \rangle) = false$ otherwise.

For details of this implementation of partitions, see *Fundamentals of data structures*, by Horowitz and Sahni, Computer Science press, pp. 248-256.

Then, the algorithm is as follows:

```

function unif[p, q, Π, dd]: flag;
  begin
    trans := left(dd); ff := right(dd); pq := (p, q); st := (pq); flag := 1;
    k := Length(first(trans));
    while st ≠ () ∧ flag ≠ 0 do
      uv := top(st); uu := left(uv); vv := right(uv);
      pop(st);
      if bad(ff, uv) = 1 then flag := 0
      else
        u := find(uu, Π); v := find(vv, Π);
        if u ≠ v then
          union(u, v, Π);
          for i = 1 to k do
            u1 := delta(trans, uu, k - i + 1); v1 := delta(trans, vv, k - i + 1);
            uv := (u1, v1); push(st, uv)
          endfor
        endif
      endif
    endwhile
  end

```

The initial partition Π is the identity relation on Q , i.e., it consists of blocks $\{q\}$ for all state $q \in Q$. The algorithm uses a stack st . We are assuming that the DFA dd is specified by a list of two sublists, the first list, denoted $left(dd)$ in the pseudo-code above, being a representation of the transition function, and the second one, denoted $right(dd)$, the set of final states. The transition function itself is a list of lists, where the i -th list represents the i -th row of the transition table for dd . The function $delta$ is such that $delta(trans, i, j)$ returns the j -th state in the i -th row of the transition table of dd . For example, we have a DFA

$$dd = (((2, 3), (2, 4), (2, 3), (2, 5), (2, 3), (7, 6), (7, 8), (7, 9), (7, 6)), (5, 9))$$

consisting of 9 states labeled $1, \dots, 9$, and two final states 5 and 9. Also, the alphabet has two letters, since every row in the transition table consists of two entries. For example, the two transitions from state 3 are given by the pair $(2, 3)$, which indicates that $\delta(3, a) = 2$ and $\delta(3, b) = 3$.

Implement the above algorithm, and test it at least for the above DFA dd and the pairs of states $(1, 6)$ and $(1, 7)$. Pay particular attention to the input and output format. Explain your data structures.

Extra Credit (up to **100 pts**). Implement your program in such a way that it displays the simultaneous parallel forward moves in the DFA, and the updating of the trees representing the blocks of the partition. You should produce a Java Applet.

TOTAL: 80 + 100 points.