Spring, 2014 CIT 596

Theory of Computation

Final recitation (solutions/ideas)

- Sipser 4.2

  The language can be expressed as this set

  $\{< D, R > |$D is a dfa and R is a regex where L(D) = L(R)$\}$

  The algorithm for deciding this language is

  1. Convert the input regular expression R to an NFA N.
  2. Convert N to a DFA D'.
  3. Use the fact that the equivalence of two DFAs is a decidable problem. Feed D and D' to a Turing Machine M that decides whether or not they are equal
     (a) If M says that D and D' are equivalent then return *accept*
     (b) If M says D and D' are not equivalent then return *reject*.

- Sipser 4.3

  There are several possible solutions for this question. One of them was discussed in the recitation where we used EQ_{DFA} as the subroutine. Here is a different solution where we will use E_{DFA} as a subroutine. Remember that E_{DFA} is the one that tells you whether the input DFA has an empty language. In other words, the question 'does a DFA accept anything at all' is decidable.

  On input $< A >$, where A is a DFA, do the following

  1. Convert A to A^c, the DFA that accepts the complement of the language being accepted by A.
  2. Using E_{DFA} as a subroutine, check and see if $L(A^c) = \phi$ or not.
  3. If $L(A^c) = \phi$ return *accept* else return *reject*.

- Sipser 4.4

  Again, there are a few different ways to do this question. Honglin described one solution using the idea of 'marking'.

  Another solution is to the following

  1. Convert G to Chomsky Normal Form. Let the collection of rules after this conversion process be represented by R_{CNF}.

2. Loop over the rules in R_{CNF}. If you find a rule $S \to \varepsilon$, then the grammar does generate ε so return *accept*. Otherwise return *reject*.

- Sipser 4.5 is solved in the book. Please read that and ask us if you have any trouble understanding.

- Sipser 4.12

  Several possible ways to do this. We will rely on three things here. That we can make a DFA that only accepts strings containing odd number of 1s. We can 'intersect' two DFAs and finally that checking the 'emptiness' of a DFA is a decidable thing.

  Formally speaking here are the steps

  1. Make D_{odd} which is a DFA that accepts only strings that have an odd number of 1s.
  2. Use the input DFA M to create a DFA D_{inter} which accepts the intersection of $L(D_{odd})$ and $L(M)$.
  3. Use E_{DFA} as a subroutine to check and see if D_{inter} accepts any string at all. If D_{inter} accepts a string then return *reject*. Else return *accept*.

- Sipser 4.13

  We know how to convert regular expressions to DFAs. Note that $A \cap B = A$ iff $A \subseteq B$.

  So we use this idea in the following steps

  1. Convert R to D_R and S to D_S.
  2. Use the intersection construction to make D_{inter}, a DFA that accepts $D_R \cap D_S$.
  3. Use EQ_{DFA} as a subroutine to check equivalence of D_{inter} and D_R. So feed $< D_{inter}, D_R >$ to EQ_{DFA}.
     - If EQ_{DFA} returns accept then return *accept*.
     - If EQ_{DFA} return reject then return *reject*.