Theory of Computation	**Lecture 10**
CS682 Spring 1997	**Tuesday, February 26, 2002**

Complexity of the Equational Theory

In this lecture we show that the equational theory of Kleene algebra is *PSPACE*-complete.

Theorem 0.1 (Stockmeyer and Meyer [2]) *The problem of deciding whether $R_\Sigma(s) = R_\Sigma(t)$ for given regular expressions s and t is PSPACE-complete. The problem is PSPACE-hard even if one of the expressions is a trivial expression representing Σ^*.*

Proof. To show that the problem is in *PSPACE*, we first convert the expressions to finite automata M and N using the construction of Kleene's theorem and ask whether the two automata accept the same set. The conversion of Kleene's theorem is linear. We will actually give a nondeterministic *PSPACE* algorithm for determining whether the two automata are different—that is, whether there exists a string accepted by one and not by the other. The nondeterminism will be used to guess the string in the symmetric difference of the two sets. By Savitch's theorem [1], nondeterministic and deterministic *PSPACE* are equivalent, so this is sufficient.

The algorithm starts with a pebble on each start state. It then guesses an input string x symbol by symbol, moving pebbles on the states of M and N to mark all states reachable from the start state under the string guessed so far. We do not need to remember the guessed string, just the states that M and N could currently be in. The procedure accepts if it ever sees a pebble configuration in which an accept state of M is occupied by a pebble, but no accept state of N is occupied by a pebble, which indicates that the string x guessed so far is accepted by M but not by N, or vice versa. Since the pebble configurations can be represented in polynomial space, this is a nondeterministic *PSPACE* computation.

We remark that in general the procedure may have to run for an exponential length of time before the acceptance situation described above occurs, if at all. This is because the shortest string *not* accepted by one of the machines may be exponential in the size of the machine. This may be surprising in light of the fact that the shortest string accepted is linear in the size of the machine, if any string at all is accepted. But consider for example an automaton over a single letter alphabet with a start state going to n disjoint loops with pairwise relatively prime lengths $p_1, \ldots, p_n$. Make every state an accept state except for the one in each loop farthest from the start state. There are $1 + p_1 + p_2 + \cdots + p_n$ states, but the shortest string not accepted is of length $p_1 p_2 \cdots p_n$.

To show that the problem is hard for *PSPACE*, we will encode computation histories of an arbitrary one-tape polynomial-space-bounded deterministic Turing machine. Let N

be such a machine, and let n^c be its space bound. Assume without loss of generality that N always erases its tape and moves its tape head all the way to the left end of the tape before accepting. Given an input x, $|x| = n$, we build a regular expression s of length $O(n^c)$ over a finite alphabet Δ such that $R_\Delta(s)$ consists of all strings that are *not* valid accepting computation histories of N on input x. Then $R_\Delta(s) = \Delta^*$ iff there is no such valid accepting computation history, which occurs iff N does not accept x. Thus we are reducing the set accepted by N to the set $\{s \mid R_\Delta(s) \neq \Delta^*\}$. The alphabet Δ, the constant c, and the constant of proportionality in the $O(n^c)$ may depend on N but are independent of x.

Formally, an *accepting computation history* of N on input x, $|x| = n$, is a string of the form

$$\#\alpha_0\#\alpha_1\#\alpha_2\# \quad \cdots \quad \#\alpha_{m-1}\#\alpha_m\# \tag{1}$$

where each α_i is a string of length n^c over a fixed finite alphabet Δ encoding a configuration of N on input x, such that:

(i) α_0 represents the start configuration of N on x

(ii) α_m represents the accept configuration of N on x

(iii) each α_{i+1} follows from α_i according to the transition rules of N.

If a string is *not* an accepting computation history, then either it is not of the form (1), or one of the three conditions (i), (ii), (iii) fails. The regular expression s will thus consist of a sum of four subexpressions s_0, s_1, s_2, and s_3, where s_0 describes those strings in Δ^* that are not of the form (1), and s_i represents the set of strings that violate condition i, $\leq i \leq 3$.

To be of the form (1), a string must be a member of the regular set denoted by $(\#\Delta^{n^c})^*\#$, plus a few other simple conditions on the format: exactly one state of N in each configuration, each configuration begins and ends with endmarkers, etc. This requires $O(n^c)$ states of M. Checking (i) or (ii) involves just checking whether the input begins or ends with a certain fixed string of length n^k. These strings are encoded in the finite control of M.

Finally, to check (iii), observe that there is a finite set of local conditions involving the $j-1^{\text{st}}$, j^{th}, and $j+1^{\text{st}}$ symbols α_i and the j^{th} symbol of α_{i+1} such that (iii) holds iff these local conditions are satisfied for all $1 \leq j \leq n^k$ and all $0 \leq i < m$. The local conditions describe all the valid transitions and depend only on the description of N. To check that (iii) is violated, M scans across the input, and at some point guesses nondeterministically where the violation occurs. It remembers the next three symbols in its finite control, skips over the next n^k symbols, and accepts if the next symbol is not correct. $\square$

References

[1] W. Savitch. Relationship between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.

[2] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Proc. 5th Symp. Theory of Computing*, pages 1–9, New York, 1973. ACM.