

- 8.28 Show that A_{NFA} is NL-complete.
- 8.29 Show that E_{DFA} is NL-complete.
- *8.30 Give an example of an NL-complete context-free language.
- A*8.31 Define $CYCLE = \{\langle G \rangle \mid G \text{ is an directed graph that contains a directed cycle}\}$. Show that $CYCLE$ is NL-complete.

SELECTED SOLUTIONS

- 8.5 Let A_1 and A_2 be languages that are decided by NL-machines N_1 and N_2 . We construct three Turing machines: N_{\cup} deciding $A_1 \cup A_2$; N_{\circ} deciding $A_1 \circ A_2$; and N_* deciding A_1^* . Each of these machines receives input w .

Machine N_{\cup} nondeterministically branches to simulate N_1 or to simulate N_2 . In either case, N_{\cup} accepts if the simulated machine accepts.

Machine N_{\circ} nondeterministically selects a position on the input to divide it into two substrings. Only a pointer to that position is stored on the work tape—insufficient space is available to store the substrings themselves. Then N_{\circ} simulates N_1 on the first substrings, branching nondeterministically to simulate N_1 's nondeterminism. On any branch that reaches N_1 's accept state, N_{\circ} simulates N_2 on the second substring. On any branch that reaches N_2 's accept state, N_{\circ} accepts.

Machine N_* has a more complex algorithm, so we describe its stages.

$N_* =$ "On input w :

1. Initialize two input position pointers p_1 and p_2 to 0, the position immediately preceding the first input symbol.
 2. *Accept* if p_2 is on the last symbol of w .
 3. Move p_2 forward to a nondeterministically selected input position.
 4. Simulate N_1 on the substring of w from the position following p_1 to the position at p_2 , branching nondeterministically to simulate N_1 's nondeterminism.
 5. If this branch of the simulation reaches N_1 's accept state, copy p_2 to p_1 and go to stage 2."
- 8.7 We construct a TM M to decide A_{DFA} . When M receives input $\langle A, w \rangle$, a DFA and a string, M simulates A on w by keeping track of A 's current state and its current head location, and updating them appropriately. The space required to carry out this simulation is $O(\log n)$, because M can record each of these values by storing a pointer into its input.
- 8.31 We reduce $PATH$ to $CYCLE$. The idea behind the reduction is to modify the $PATH$ problem instance $\langle G, s, t \rangle$ by adding an edge from t to s in G . If a path exists from s to t in G , a directed cycle will exist in the modified G . However, other cycles may exist in the modified G because they may already be present in G . To handle that problem, we first change G so that it contains no cycles. A **leveled directed graph** is one where the nodes are divided into groups, A_1, A_2, \dots, A_k , called *levels*, and only edges from one level to the next higher level are permitted. Observe that a leveled graph is acyclic. The $PATH$ problem for leveled graphs is still NL-complete, as the following reduction from the unrestricted $PATH$ problem shows. Given a graph G with two nodes s and t , and m nodes in total, produce the leveled graph G' whose

levels are m copies of G 's nodes. Draw an edge from node i at each level to node j in the next level if G contains an edge from i to j . Additionally, draw an edge from node i in each level to node i in the next level. Let s' be the node s in the first level and let t' be the node t in the last level. Graph G contains a path from s to t iff G' contains a path from s' to t' . If we modify G' by adding an edge from t' to s' we obtain a reduction from *PATH* to *CYCLE*. The reduction is computationally simple, and its implementation in logspace is routine. Furthermore, a straightforward procedure shows that *CYCLE* \in NL. Hence *CYCLE* is NL-complete.