Fall 2023

Review problems on Space Complexity

Solutions

1. Consider the decision problem
   DFA-ACCEPT: *Given a DFA* M *over an alphabet* Σ *and a* Σ*-string* w *does* M *accept* w.

   The problem **NFA-ACCEPT** is defined similarly, referring to nondeterministic automata (NFAs).

   (a) Show that **DFA-ACCEPT** is decidable in linear time.

   (b) Show that **NFA-ACCEPT** is decidable in linear space.

   Argue informally, referring to transition diagrams.

   Solution.

   ► An instance M, w of DFA-ACCEPT can be decided by an on-site acceptor that runs M on w. The simulation is in time linear in w, since a DFA consumes one symbol of its input at each computation step. But with respect to the combined input M, w, the simulation is in time $|M| \times |w|$, since every move of M requires a scan of M to detect the applicable transition.

   ► An instance N, w of **NFA-ACCEPT** can be decided by an on-site acceptor that reads w and successively updates the set of states reached by reading the initial substring w read so far, as in the conversion of an NFA to a DFA. Each updates requires only a scan of the transition table of M and no additional space.

2. A ***string-ladder*** over an alphabet Σ is a sequence $w_1, \ldots, w_k \in \Sigma^n$ of equal-length strings (separated by commas), where each w_{i+1} differs from w_i in exactly one letter. For example, here is a string-ladder of English words, with commas used as a separator between entries: `near`, `fear`, `feat`, `beat`, `best`, `vest`, `vast`.

   Show that the set of string-ladders over Σ^* , is in $\mathbf{L}$ (i.e. log-space decidable). To do this, describe an algorithm implementable on a multi-cursor two-way automaton, that recognizes the string-ladders.

   Solution. On input w the algorithm

   (a) Steps one cursor forward until it finds a comma, and places a second cursor at the first symbol.

(b) It then repeats until the lead cursor reaches then end of the input:
scan adjacent strings with the two cursors stepping in tandem. Proceed if the two strings differ by exactly one symbol, abort otherwise.

(c) Accept if and when the loop above terminates without aborting.

3. Consider the CFL B consisting of balanced parentheses, such as (( )( )) but not (( ))(. Show that B is in **L**. [Hint: Think of counting the excess of left parentheses over right parentheses. A string of parentheses is balanced iff that count is never negative, and it gets to 0 at the end of the input.]

Solution. An acceptor for B is obtained by counting on the work-string, in binary, the excess of the count of left parentheses over the count of right parentheses. If a right parenthesis is encountered when the count is 0, abort. The input is accepted if the count is 0 at the end of the input.

For input of size n the count cannot exceed n, so the binary counters have length $\leqslant \log n$.

4. Show that the following decision-problem is in **PSpace**.
DFA-EMPTINESS: *Given a DFA M does it accept some string?*
(I.e., is $\mathcal{L}(M) \neq \emptyset$?)

[Hint: How long is the shortest string accepted by a DFA with k states? It is easy to show that the problem is in co-NP, from which PSpace easily follows.]

Solution. If a k-state DFA M accepts a string w of length $n > k$ then the Clipping Theorem applies to w and M accepts some string of length $p < n$.

Thus a k-state instance M of DFA-EMPTINESS recognizes a non-empty language iff M accepts a string w of length $\leqslant k$. It suffices therefore to cycle through all such strings w and check, in space linear in $M\square w$, whether M accepts w. Each such cycle reuses the space occupied by w, and $|w| \leqslant k \leqslant |M|$, so the algorithm is in linear space.

5. We say that boolean expressions E and F are ***equivalent*** if they have the same truth table; that is, msE and F use the same set number of variables and return the same truth value for each valuation.

Show that the following problem is in **PSpace**:
BOOL-EQUIV: *Given two boolean expressions, are they equivalent?*
[Hint: Show that EXP-EQUIV is Co-NP, then use **NP** $\subseteq$ **PSpace** and Problem 1.]

Solution. The complement problem EXP-NONEQUIV is **NP**:
the witness for the non-equivalence of expressions E and F s a valuation that yields 0 for one of the expressions and 1 for the other. Thus EXP-EQUIV is **co-NP**. But since **NP** $\subseteq$ **PSpace** , So **coNP** $\subseteq$ **co-PSpace** = **PSpace**.

6. An expression msE is ***minimal*** if there is no shorter expression equivalent to it.

   Show that the following problem is in **PSpace**:
   MIN-BOOL: *Given a boolean expression, is it minimal?*

   Solution. Consider the following decision algorithm: Given an instance F of **MIN-BOOL** cycle through all shorter expressions G, checking for each whether it is equivalent to F. The space required for checking equivalence is PSpace in the size of (F, G), by the previous problem, space is reused for each cycle, and the expressions G considered are of size $\leqslant |F|$. So the entire algorithm uses space polynomial in the size of F.

7. Show that if **BOOL-SAT** were **PSpace**-hard then **PSpace** = **NP**.

   Solution. We already know that **NP** $\subseteq$ **PSpace**.
   If **BOOL-SAT** were **PSpace**-hard, i.e. for each **PSpace** problem $\mathcal{P}$ is $\leqslant_p$ **bool-sat** then $\mathcal{P}$ would be **NP**, because **NP** is closed under $\leqslant_p$.

8. If M is an automaton (DFA), a string-ladder (as defined in Problem 3) is an ***M-ladder*** if each pair w_i, w_{i+1} in the ladder is accepted by M.

   Show that for each DFA M the set of M-ladders over Σ is in **PSpace**.

9. Let D be the CFL consisting of balanced parentheses and brackets, such as ([( )( )])[ ] but not ([ )]. Show that D is in **L**. [Hint: Use a multi-cursor two-way automaton to recognize D, then invoke Hartmanis's Theorem.]

   Solution. Given an input string w we make a first pass through w to determine the nesting-depth of parens and brackets, by counting the excess of left parens/brackets over right ones, and keeping record of the maximum count d (the depth of the potential parse tree of w). We have $d \leqslant |w|$ so the count is in log-space.

   (We may abort the count if it gets below 0, but note that this does not guarantee a balanced expression: The count for ([ )], which should be rejected, is never negative.))

   For each $i = 1 \ldots d$ we make two passes through w one for parens and one for brackets. The pass for parens maintains a count of the excess of right parens of depth i over left parens of depth i (depth being counted with respect to both parens and brackets). That excess should never be negative, and should reach 0 at the end of w. The pass for brackes is similar.

   For example, ([ )] will be rejected, because [is of depth $i = 2$ but] is of depth $i = 1$, so they are not matched in the same pass.