Exercise Sheet 5: Space Complexity

David Carral

December 5, 2019

Exercise 1

Let $\mathbf{A}_{\text{LBA}}$ be the word problem of deterministic linear bounded automata. Show that $\mathbf{A}_{\text{LBA}}$ is $\textsc{PSpace}$-complete.

$$\mathbf{A}_{\text{LBA}} = \{\langle \mathcal{M}, w\rangle \mid \mathcal{M} \text{ is a deterministic LBA and } w \in \mathbf{L}(\mathcal{M})\}$$

Definition.

1. A *linear bounded automata* (DLBA) is a tuple $\langle Q, \Sigma, \Gamma, \delta, q, Q_F, E_L, E_R\rangle$ where
   - Q is a set of states, Σ is the input alphabet, Γ is the tape alphabet, δ is the transition function, q is the start state, Q_F is the set of final states, and
   - E_L **and** E_R **are the left and right end markers**, respectively.
2. For all $\langle q, a\rangle \mapsto \langle q, b, M\rangle \in \delta$ (where $q \in Q$, $a, b \in \Gamma$, $M \in \{R, L\}$),
   - if $a = E_L$, then $b = E_L$ and $M = R$, and
   - if $a = E_R$, then $b = E_R$ and $M = L$.
3. Inputs are of the form $E_L, w_1, \ldots, w_n, E_R$ with $w_i \notin \{E_L, E_R\}$ for all $i \in \{1, \ldots, n\}$.

Exercise 1

Let $\mathbf{A}_{\text{LBA}}$ be the word problem of deterministic linear bounded automata. Show that $\mathbf{A}_{\text{LBA}}$ is PSPACE-complete.

$$\mathbf{A}_{\text{LBA}} = \{\langle \mathcal{M}, w \rangle \mid \mathcal{M} \text{ is a deterministic LBA and } w \in \mathbf{L}(\mathcal{M})\}$$

Solution. Show that $\mathbf{A}_{\text{LBA}}$ is in PSPACE.

- A DLBA $\mathcal{M}$ on input w may traverse at most $|Q| \cdot |w| \cdot |\Gamma|^{|w|}$ different configurations.
- To decide whether $\mathcal{M}$ terminates on w, we simulate this LBA on w for $|Q| \cdot |w| \cdot |\Gamma|^{|w|}$ steps. If the simulation accepts, we *accept*. Otherwise, we *reject*.
- To track the number of steps, we make use of a binary counter that takes at most $\log|Q| + \log|w| + |w| \cdot \log|\Gamma|$ cells.

Exercise 1

Let $\mathbf{A}_{\text{LBA}}$ be the word problem of deterministic linear bounded automata. Show that $\mathbf{A}_{\text{LBA}} = \{\langle \mathcal{M}, w \rangle \mid \mathcal{M}$ is a (deterministic) LBA and $w \in \mathbf{L}(\mathcal{M})\}$ is $\textsc{PSpace}$-complete.

Solution. To show that $\mathbf{A}_{\text{LBA}}$ is $\textsc{PSpace}$-hard, we provide a reduction from a $\textsc{PSpace}$-hard problem into $\mathbf{A}_{\text{LBA}}$. Namely, we reduce **QBF** into $\mathbf{A}_{\text{LBA}}$.

- Let $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q, Q_F \rangle$ be some TM that (i) solves **QBF** and (ii) uses at most $p(|w|)$ cells on input w with p a polynomial function.
- Let $\mathcal{M}' = \langle Q, \Sigma, \Gamma, \delta', q, Q_F, E_L, E_R \rangle$ with δ' the smallest superset of δ.
- Let f be a function mapping every QBF formula $\langle w \rangle$ onto $E_L \langle \mathcal{M}', w \rangle \mathrm{B}^k E_R$ where B is the blank symbol and $|\langle w \rangle| + k = p(|\langle w \rangle|)$.
- The function f is a poly-time reduction from **QBF** into $\mathbf{A}_{\text{LBA}}$ (discuss).

Exercise 2

Consider the Japanese game *go-moku* that is played by two players X and O on a 19x19 board. Players alternately place markers on the board, and the first one to have five of its markers in a row, column, or diagonal wins. Consider the generalised version of go-moku on an $n \times n$ board. Say that a *position* of go-moku is a placement of markers on such a board as it could occur during the game, together with a marker which player moves next. We define

$$\textbf{GM} = \{\langle B\rangle \mid B \text{ is a position of go-moku where X has a winning strategy}\}.$$

Show that **GM** is in $\mathrm{PS{\scriptsize PACE}}$.

Exercise 2

Let **GM** $= \{\langle B\rangle \mid B$ is a position of go-moku where X has a winning strategy$\}$.

Solution. To show that **GM** is in PSPACE we provide an algorithm that solves **GM** assuming that we receive a valid position as input.

```
boolean $XHasWinStr(\langle B\rangle)$ {
        if($isXWinPos(\langle B\rangle)$) return $true$;
        if($isOWinPos(\langle B\rangle) \vee FullBoard(\langle B\rangle)$) return $false$;
        if ($IsXTurn(\langle B\rangle)$)
            for each $\langle B'\rangle \in next(\langle B\rangle)$
                if($XHasWinStr(\langle B'\rangle)$) return $true$;
            return $false$;
        else
            for each $\langle B'\rangle \in next(\langle B\rangle)$
                if($\neg XHasWinStr(\langle B'\rangle)$) return $false$;
            return $true$; }
```

Remarks:

1. Check correctness of the starting position.
2. The depth of the recursion is n^2 (compare with chess).
3. Encoding each position takes $O(n^2)$ space.

Exercise 3

Show that the universality problem of nondeterministic finite automata $\textbf{ALL}_{\text{NFA}} = \{\langle \mathcal{A} \rangle \mid \mathcal{A} \text{ an NFA accepting every valid input}\}$ is in $\textsc{PSpace}$.

Solution. Preliminary argument.

1. Let $\mathcal{B}$ be some NFA with n states such that $L(\mathcal{B}) \neq \Sigma^*$.
2. By (1), there is some DFA $\mathcal{B}'$ with at most 2^n states such that $L(\mathcal{B}) = L(\mathcal{B}')$.
3. By (1) and (2), $L(\mathcal{B}') \neq \Sigma^*$. That is, there is a non-final state in $\mathcal{B}'$ that is reachable from the initial state of $\mathcal{B}'$.
4. By (3), there is some $w \in \Sigma^*$ with $|w| \leq 2^n$ and $w \notin L(\mathcal{B})$.
5. By (1) and (4), if $\mathcal{B}$ is an NFA with n states and $L(\mathcal{B}) \neq \Sigma^*$, then there is some $w \in \Sigma^*$ such that $|w| \leq 2^n$ and $w \notin L(\mathcal{B})$.

Exercise 3

Show that the universality problem of nondeterministic finite automata $\mathbf{ALL}_{\mathsf{NFA}} = \{\langle\mathcal{A}\rangle \mid \mathcal{A} \text{ an NFA accepting every valid input}\}$ is in $\mathrm{PS{\small PACE}}$.

Solution. Step by step proof.

1. Let $\mathcal{A}$ be some NFA with n states.
2. Then, we construct a NDTM $\mathcal{M}$:
   - We can guess one after the other letters from the input alphabet and we simulate a run of $\mathcal{A}$ on the corresponding input.
   - We repeat this guessing up until 2^n steps (this can be done using a counter, which only requires $O(n)$ space).
   - If after at most 2^n steps $\mathcal{A}$ does accept, we *accept*; otherwise, we *reject*.
3. $\mathcal{M}$ uses polynomial space and $L(\mathcal{M}) = \overline{\mathbf{ALL}_{\mathsf{NFA}}}$.
4. By (3), $\overline{\mathbf{ALL}_{\mathsf{NFA}}} \in \mathrm{NPS{\small PACE}}$.
5. By (4), $\overline{\mathbf{ALL}_{\mathsf{NFA}}} \in \mathrm{PS{\small PACE}}$.
6. By (5), $\mathbf{ALL}_{\mathsf{NFA}} \in \mathrm{PS{\small PACE}}$.

Exercise 4

Show that the composition of logspace reductions again yields a logspace reduction.

Solution. Set up.

1. Let $f, g\colon \Sigma^* \to \Sigma^*$ be logspace-computable functions.
2. Let $\mathcal{M}_f$ and $\mathcal{M}_g$ be logspace transducers computing f and g, respectively.
   Definition: A *log-space transducer* $\mathcal{M}$ is a logarithmic space bounded Turing machine with a read-only input tape and a write-only, write-once output tape, and that halts on all inputs.
3. To show: $f \circ g$ is also logspace-computable.

Naive (and incorrect) solution:

1. Consider the composition of $\mathcal{M}_f$ and $\mathcal{M}_g$: on input w we first compute $g(w)$ using $\mathcal{M}_g$, and then $f(g(w))$ using $\mathcal{M}_f$.
2. Problem: even though $\mathcal{M}_g$ uses only logarithmic space on its working tape, the output $g(w)$ can be polynomially large in the size of w!
3. Hence, $\mathcal{M}_f$ may not have enough memory to store $g(w)$.

Exercise 4

Show that the composition of logspace reductions again yields a logspace reduction.

Solution.

1. We can modify $\mathcal{M}_g$ to compute only the k-th symbol of $g(w)$ on demand. This can be done using enhancing $\mathcal{M}_g$ by a counter p that indicates the k-th symbol of $g(w)$.
2. A logspace transducer computing $f(g(w))$ now works as follows.
   - We start by simulating the computation of $\mathcal{M}_f$.
   - Every time this simulation wants to read a symbol from $g(w)$ at position q, we use the modified version of $\mathcal{M}_g$ to compute this symbol
   - Both the simulation of $\mathcal{M}_f$ as well as the on-demand computation of $\mathcal{M}_g$ can be done in logarithmic space, and hence, we can compute $f(g(w))$ in logarithmic space.
3. **Remark.** The size of the output of $\mathcal{M}_g$ on input w is $k = |\Gamma|^{p(|w|)} \cdot p(|w|) \cdot |Q|$ where p is a logarithmic function, and Γ and Q are the tape alphabet and the set of states of $\mathcal{M}_g$, respectively. To store such a number, we can use a binary counter that takes $\log(k) = p(|w|) \cdot \log(|\Gamma|) + \log(p(|w|)) + \log(|Q|)$ cells.

Exercise 5

Show that the word problem $\mathbf{A}_{NFA}$ of non-deterministic finite automata is NL-complete.

Solution. We show $\mathbf{A}_{NFA} \in \mathrm{NL}$.

1. Let $\mathcal{A}$ be an NFA and w some valid input word.
2. Without loss of generality, we assume that $\mathcal{A}$ does not contain ε transitions (see Exercise 4).
3. We can simply simulate $\mathcal{A}$ on input w. For this we only need to store two pointers:
   - One indicating the current position in w.
   - One pointing to the current state of $\mathcal{A}$.
4. For each transition of $\mathcal{A}$ with more than one possibility we make a guess.
5. When we have reached the end of the input w the machine accepts if and only if the current state of $\mathcal{A}$ is a final state.

Exercise 5

Show that the word problem $\mathbf{A}_{NFA}$ of non-deterministic finite automata is NL-complete.

Solution. To show that $\mathbf{A}_{NFA}$ is NL-hard, we provide a logspace reduction from reachability in directed graphs.

1. Let G be a directed graph and let s, t be vertices in G.
2. Let $\mathcal{A}_G$ be the NFA such that
   - $u \xrightarrow{\varepsilon} v \in \mathcal{A}_G$ if $u \to v \in G$,
   - the only initial state of $\mathcal{A}_G$ is s, and
   - the only final state of $\mathcal{A}_G$ is t.
3. G has a path from s to t if and only if $\mathcal{A}_G$ accepts the empty word.
4. $\mathcal{A}_G$ can be built in logarithmic space.

Exercise 6

Show that **BIP** $= \{ \langle G \rangle \mid G$ a finite bipartite graph $\}$ is in NL.

Solution. We first show that, a graph G is bipartite if and only if it does not contain any cycles of odd length.
If G does not contain any cycles of odd length, then it is bipartite.

1. Let G be a graph without cycles of odd length.
2. Choose some vertex v in G, colour it with blue, and then colour all of its neighbours with red.
3. Then, all the neighbours of red vertices are coloured blue again.
4. Iteratively apply (2) and (3) until every vertex is coloured.
5. As G only contains cycles of even length, we never colour the same vertex with both colours.
6. Place blue vertices in one set, and red in the other.

If a graph G is bipartite, then it does not contain any cycles of odd length.

- Contrapositive: If G contains a cycle of odd length, then it is not bipartite.

Exercise 6

Show that **BIP** $= \{ \langle G \rangle \mid G$ a finite bipartite graph $\}$ is in NL.

Solution. We show that $\overline{\textbf{BIP}} \in$ NL:

1. Shown on the previous slide: a graph is not bipartite iff it contains an odd cycle.
2. Let $\mathcal{M}$ be the NDTM that performs the following computation on input graph G.
   - Non-deterministically select a vertex v from G and some odd number $\ell \leq |E|$.
   - Then, traverse a path of length ℓ and checks if it ends with v.
   - The algorithm *accepts* if this is the case, and *rejects* otherwise.
3. The TM M only has to store the vertex v, the current vertex that is being traversed during the computation of the path, and the number of vertices still to be visited (doable in logarithmic space).

Because NL $=$ coNL, we obtain **BIP** $\in$ NL, as required.