Exercise 1 : Functions computable in logspace

Let $A$ be some finite alphabet. With a partial function $f : A^* \rightarrow A^*$, defined on $\text{dom } f$, we associate the following language

$$D_f = \{ (x, i, a) \in A^* \times \mathbb{N} \times A \mid x \in \text{dom } f \text{ and } 0 < i \leq |f(x)| \text{ and } a \text{ is the } i\text{-th letter in } f(x) \}.$$

For representation purposes, we assume that the number $i$ is written in binary and that $D_f$ uses a new symbol, say $\# \notin A$, to separate the components of a triple. Thus $D_f \subseteq A^*$ for $A \overset{def}{=} A \cup \{0, 1, \#\}$.

Recall that $f$ is logspace computable iff there exists a deterministic Turing machine $M_f$ that, when started with some $x \in A^*$ on its (bidirectional, read-only) input tape, terminates, produces $f(x)$ on its (unidirectional, write-only) output tape, and only uses $O(\log |x|)$ cells on its work tapes. When $f(x)$ is undefined, $M_f$ terminates by reaching a rejecting/error/.. state.

1. Prove that a total function $f$ is logspace computable if, and only if, $D_f \in L$ and there exists a polynomial $p$ such that $|f(x)| \leq p(|x|)$ for all $x \in \text{dom } f$.

Solution:

($\Rightarrow$) As seen in class, if a deterministic TM is logspace it can only, when started on some input $x$, visit a polynomially-bounded number of configurations before halting, hence $|f(x)|$ is polynomially-bounded. Furthermore, deciding whether $(x, i, a) \in D_f$ can be done by simulating $M_f$ on $x$ and, instead of writing $f(x)$ on the output tape, we maintain a counter $c$ that keeps track of how many characters would have been output. When $c$ coincides with $i$, we can compare $a$ with the character that would be output by $M_f$. If the simulation terminates before $c$ equals $i$, we deduce that $|f(x)| < i$ (or $x \notin \text{dom } f$) and we can reject $(x, i, a)$.

In fine, this uses the same work space as $M_f$ plus logspace overhead for $c$, hence is in $L$.

($\Leftarrow$) : Given a logspace Turing machine $M$ deciding $D_f$, we can produce a machine $M'$ computing $f$ in logspace : $M'$ computes $p(|x|)$ and then loops over all $i = 1, \ldots, p(|x|)$ and all $a \in A$, simulating $M$ on $(x, i, a)$. If the triple is accepted, $M'$ outputs $a$ and goes to next $i$. When there is no $a \in A$ with $(x, i, a) \in D_f$, we know that $i$ is larger than $|f(x)|$ hence we have finished outputting $f(x)$ and we can accept. This uses logspace extra workspace (for computing $p(|x|)$, for looping over $i$ and $a$) on top of what $M$ uses, hence is in logspace.

2. In question 1, can we remove the assumption on $p$, i.e., do we also have “$D_f \in L$ iff $f$ is logspace computable”?

Solution:

The answer is no since $D_f \in L$ does not imply that $f(x)$ has a polynomially-bounded length. For example consider $A = \{a\}$ and $f(x) = a^{2^x}$ for all $x \in A^*$. Having exponential length, $f(x)$ is not
computable in polynomial time, nor a fortiori in logspace. However $D_f$ is in $L$: one has

$\langle x, i, a \rangle \in D_f \text{ iff } 1 \leq i \leq 2^{|x|}$

and, given some $\langle x, i, a \rangle$, testing whether $i \leq 2^{|x|}$ reduces to comparing the length of (the encoding of) $i$ and the length of $x$: this can certainly be done in logspace.

3. In question 1, can we remove the assumption that $f$ is a total function?

**Solution:**
No. Take any $P \subseteq A^*$ and define $f$ by $f(x) = \epsilon$ if $x \in P$ and $f(x)$ undefined otherwise. Then $D_f$ is empty and $|f(x)| = 0$ for all $x \in \text{dom } f = P$, hence $D_f$ is in $L$ and $|f(x)|$ is polynomially-bounded. However $f$ is not computable when $P$ is not decidable.

**Exercise 2 : Closure via operations**

We consider the following decision problem:

**Problem :** BinOpGen

**Input :** A finite set $X$, a binary operation $*: X \times X \rightarrow X$, a subset $S \subseteq X$ and a target $t \in X$.

**Question :** Does $\langle S \rangle$ denote the closure of $S$, a set of generating elements, by $*$. Formally, we define an increasing sequence $S_0, S_1, S_2, \ldots$ of subsets of $X$ with

$S_{i+1} = S_i \cup \{ x * y \mid x, y \in S_i \}$

and let $\langle S \rangle = \bigcup_{i \in N} S_i$.

For representation purposes, we may assume that $X$ is a finite set of the form $\{1, \ldots, n\}$, so that $*$ can be represented as an $n \times n$ matrix with values in $\{1, \ldots, n\}$.

A related problem is BinOpGen, where we consider the closure of $S$ via two binary operations on $X$, say $*_1$ and $*_2$. Formally, we define $\langle S \rangle_{*1,*2}$ as $\bigcup_{i \in N} S_{i,*1,*2}$, replacing $(\dagger)$ with

$S_{i+1,*1,*2} = S_{i,*1} \cup \{ x *_1 y \mid x, y \in S_{i,*1,*2} \}$

and again asking whether $t \in \langle S \rangle_{*1,*2}$.

4. Show that BinOpGen and BinOpGen are inter-reducible, hence “have the same complexity”.

**Solution:**

The reduction BinOpGen $\leq$ BinOpGen is easy : we duplicate $*$ and leave $X, S, t$ unchanged.

For $\leq$ BinOpGen, a possible solution is to define

$X',*'_1,*'_2,S,t \mapsto X',*'_1,*'_2,S',t'$

with $X' = X \times \{1, 2\}$ and $*'_1,S',t'$ given by

$(x,k) *'_1(y,l) = (x *_k y,l), \quad S' = S \times \{1, 2\}, \quad t' = (t, 1)$

We claim that $S'_{i,*1,*2} = S_{i,*1,*2} \times \{1, 2\}$ for all $i \in N$ and prove this by induction on $i$ (easy proof omitted). Finally, $t \in \langle S \rangle_{*1,*2}$ iff $(t,1) \in \langle S' \rangle_{*1,*2}$ so the reduction is correct. That it is logspace clear since outputting $*$ just needs two nested loops on $X$, and $X', S'$ are even easier to produce.

TernOpGen is a further variant where we are given a ternary operation, denoted $\phi$, over $X$, and where one asks, given $X, \phi, S, t$, whether $t \in \langle S \rangle_{\phi}$. Here $(\dagger)$ is replaced by $S_{i+1,\phi} = S_{i,\phi} \cup \{ \phi(x,y,z) \mid x,y,z \in S_{i,\phi} \}$.

5. Show that BinOpGen and TernOpGen are inter-reducible.

**Solution:**


Exercise 3: Complexity of some closure operations

We let BinOpGen be the restriction of TernOpGen to the case where * is associative, i.e., satisfies \( x * (y * z) = (x * y) * z \) for all \( x, y, z \in X \).

6. Show that GAP \( \leq \) BinOpGen_assoc, where GAP is the Graph Accessibility Problem seen in class.

Solution:
We consider a reduction
\[
G = (N, E), s, t \mapsto X, *, S, (s, t)
\]
where \( X \) is defined as \( N^2 \cup \{ \perp \} \) and with \(*\) given by, for any \( x \in X \) and any \( n_1, n_2, n_3, n_4 \in N \):
\[
(n_1, n_2) * (n_3, n_4) = \begin{cases} 
(n_1, n_4) & \text{if } n_2 = n_3, \\
\perp & \text{otherwise},
\end{cases}
\]

We observe that, as required, this indeed defines a transitive * (where \( \perp \) is absorbing). We further let \( S \) be the restriction ofBinOpGen to the case where \( \perp \) is absorbing. We now claim that, for all \( n, n' \in N \) and \( i \in N \), \((n, n') \in S_i \) if \( G \) has a path \( n \rightarrow n' \) of length \( \leq 2^i \). This is easily proven by induction on \( i \). As a consequence, \((s, t) \in (S)_* \) if \( G \) has a path \( s \rightarrow t \), hence the reduction is correct.

The reduction is logspace since producing * only requires four nested loops on \( N \), while \( X \) and \( S \) are even easier to produce.

7. Show that BinOpGen_assoc is NL-complete.

Solution:
BinOpGen_assoc is NL-hard as shown in the previous question, and there only remains to show that the problem can be solved in NL. For this, and thanks to associativity, we observe that, given an instance \((X, *, S, t)\), it is enough to guess a sequence \( n_1, \ldots, n_k \) of elements of \( S \) and check that \( t = n_1 * n_2 \cdots n_k \).

Let us now prove that when such a sequence exists, the shortest one has length \( k \leq |X| \); given a sequence \( n_1, \ldots, n_k \) generating \( t \), we write \( m_1 \) for the partial product \( n_1 * n_2 * \cdots * n_{\ell} \). If \( \ell > |X| \) then two partial products coincide, say \( m_i = m_j \) for some \( 1 \leq i < j \leq k \). Then \( t = (n_1 * \cdots * n_i) * (n_{j+1} * n_{j+2} * \cdots * n_k) \) and there is a shorter way of generating \( t \).

It is now easy to solve BinOpGen_assoc in NL since we can guess a sequence of length at most \( |X| \) and check that its product equals \( t \). We note that we only store in memory the current partial product (and a counter \( i = 1, \ldots, k \)) while we guess the sequence. We also need to check the associativity of * before accepting an instance \((X, *, S, t)\) but this only uses three nested loops on \( X \).
We recall that \texttt{MonotoneCircuitValue} is defined as follows.

\textbf{Problem : \texttt{MonotoneCircuitValue}}

\textbf{Input :} A circuit \(C\) and one of its nodes \(n_f\).

\textbf{Question :} Does \(v_C(n_f) = \text{true}\)?

A circuit is defined as an acyclic directed graph \(C = (N, E)\) with two kind of nodes: conjunctive and disjunctive (we write \(N = N_\wedge \cup N_\vee\)). The boolean value of a node \(n \in N\), written \(v_C(n)\), is defined by

\[
v_C(n) \overset{\text{def}}{=} \begin{cases} \bigwedge \{v_C(m) \mid (m, n) \in E\} & \text{if } n \in N_\wedge, \\ \bigvee \{v_C(m) \mid (m, n) \in E\} & \text{if } n \in N_\vee. \end{cases}
\]

Since the circuit is acyclic, the above definition is well-founded.

Observe that, for nodes with no inputs, the definition relies on \(\bigvee \emptyset = \text{false}\) and \(\bigwedge \emptyset = \text{true}\). We further recall that it is easy to reduce \texttt{MonotoneCircuitValue} to a restricted version, called \(\texttt{MonotoneCircuitValue}_{\text{deg}^2}\), where we only allow circuits where every node has only 2 (or 0) inputs.

8. Give a reduction witnessing \(\texttt{MonotoneCircuitValue}_{\text{deg}^2} \leq L\) where \(L\) is either BinOpGen, TernOpGen and 2BinOpGen.

\textbf{Indication :} Choose the problem that makes the reduction cleanest and easiest to prove correct. (There is no loss of generality since, as per Exercise 2, the three versions are equivalent.)

\textbf{Solution:} It will be useful to define inductively a height \(h(n) \in \mathbb{N}\) for each node in \(N\) via \(h(n) = \max\{1 + h(n') \mid (n', n) \in E\}\) (which ensures \(h(n) = 0\) if \(n\) is a leaf node with no inputs).

We define a reduction witnessing \(\texttt{MonotoneCircuitValue}_{\text{deg}^2} \leq \text{BinOpGen}\)

\[
N_\wedge, N_\vee, E, n_f \mapsto X, *, S, t
\]

with \(X \overset{\text{def}}{=} (N \cup E) \times \{?, \top\}\). Here \(X\) has two copies of each node \(n\) and of each edge \(e = (n', n'')\) of \(C\). We define \(*\) via

\[
(n, \top) * ((n', n''), ?) \overset{\text{def}}{=} ((n', n''), \top) \text{ if } n = n', \\
((n', n''), \top) * (n, ?) \overset{\text{def}}{=} (n, \top) \text{ if } n'' = n \text{ and } n \in N_\vee, \\
((n_1, n_2), \top) * ((n_3, n_4), \top) \overset{\text{def}}{=} (n_2, \top) \text{ if } n_2 = n_4 \in N_\wedge \text{ and } n_1 \neq n_3, \\
x * y \overset{\text{def}}{=} x \text{ if none of the above applies.}
\]

The above definition read informally as

\(R1\) : if we know that \(v_C(n) = \top\) and \(e\) is an edge leaving from \(n\) then we know that \(e\) carries \(\top\).

\(R2\) : if we know that edge \(e\) carries \(\top\) and is input to a conjunctive node \(n\), then \(v_C(n) = \top\).

\(R3\) : if we know that edges \(e_1\) and \(e_2\) carry \(\top\) and they are the input of a conjunctive node \(n\) then \(v_C(n) = \top\).

We further set \(S \overset{\text{def}}{=} (N \cup E) \times \{?, \top\} \cup \{(n, \top) \mid n \in N_\wedge \text{ and } h(n) = 0\}\) and claim that, for all \(i \in \mathbb{N}\) :

\[
(n, \top) \in S_{i,*} \text{ iff } v_C(n) = \text{true and } i \geq 2 \cdot h(n), \\
((n, n'), \top) \in S_{i,*} \text{ iff } v_C(n) = \text{true and } i \geq 2 \cdot h(n) + 1.
\]

This is easily proven by induction on \(i\) (proof omitted). Finally, we set \(t \overset{\text{def}}{=} (n_f, \top)\) and have \(t \in \langle S \rangle_*\) iff \(v_C(n_f) = \text{true}\) and the reduction is correct. It is easily seen to be in logspace.

9. Show that BinOpGen is PTIME-complete.

\textbf{Solution:} After the previous question, there only remains to show that BinOpGen is decidable in PTIME. For this we note that each \(S_{i,*}\) has size at most \(|X|\) and can be computed in quadratic time from \(S_{i-1,*}\) and \(*\). Finally the fixpoint \(\langle S \rangle_* = \bigcup_{i \in \mathbb{N}} S_{i,*} \) is reached after polynomially-many iterations since \(\langle S \rangle_* = S_{\ell,*}\) for \(\ell = \lvert X \rvert - 1\).