3. Logarithmic Space

Reinhard Pichler
Institute of Logic and Computation
DBAI Group
TU Wien
13 March, 2018

Example

Log-space computations
If a Turing machine with input and output is supposed to operate within space bound \( O(\log(n)) \), then it may never copy a substantial portion of the input onto its worktapes. Access to the input and control of the machine is based on the following log-space functionalities:

- maintaining (a constant number of) counters
- maintaining (a constant number of) “pointers” to the input
- “storing” a constant-size fragment of the input in the state

Example

The language of palindromes (over an arbitrary alphabet \( \Sigma \)) can be recognized by a 3-string Turing machine \( M \) with input within space bound \( O(\log(n)) \). (There is no output tape needed.)
Log-Space Reductions

Reductions in the "Formale Methoden" lecture

- 2 kinds of reductions: Turing reductions vs. many-one reductions
- Limit on the resources needed by a reduction: polynomial time vs. logarithmic space reductions.
- Default for problem reductions in "Formale Methoden" (e.g., in NP-completeness proofs): polynomial time, many-one reductions.
- In this course: We also want to prove completeness results for classes below NP (in particular, completeness in P or NL).
  - \( \implies \) We need reductions in a complexity class below P.
- From now on: log-space, many-one reductions, denoted as \( \leq_1 \).
- Remark. All polynomial-time reductions encountered so far (in the "Formale Methoden" lecture) also work in log-space!

Composing Reductions

In the "Formale Methoden" lecture, we have established the following chain of reductions:

\[ 3\text{-SAT} \leq_1 \text{INDEPENDENT SET} \leq_1 \text{VERTEX COVER} \]

But do reductions compose, i.e., is \( \leq_1 \) transitive? For instance, does \( 3\text{-SAT} \leq_1 \text{VERTEX COVER} \) hold?

Theorem

If \( R \) is a reduction from language \( L_1 \) to \( L_2 \) and \( R' \) is a reduction from \( L_2 \) to \( L_3 \), then the composition \( R \circ R' \) is a reduction from \( L_1 \) to \( L_3 \).

- As \( R, R' \) are reductions, \( x \in L_1 \iff R(x) \in L_2 \iff R'(R(x)) \in L_3 \).
- It remains to show that \( R'(R(x)) \) can be computed in \( O(\log n) \) space where \( n = |x| \).

Logarithmic space consumption (continued)

- Initially \( i = 1 \) and it is easy to simulate the first move of \( M_R \) (scanning \( \triangleright \)).
  - If \( M_R \) moves the cursor on the input tape to the right, then simulate \( M_R \) to generate the next output symbol and increment \( i \) by one.
  - If \( M_R \) moves the cursor on the input tape to the left, then decrement \( i \) by one and run \( M_R \) on \( x \) from the beginning, counting the output symbols and stopping when the \( i \)-th symbol is output.
  - The space required for simulating \( M_R \) on \( x \) as well as \( M_{R'} \) on \( Rx \) is \( O(\log n) \) where \( n = |x| \).
  - The space needed for bookkeeping the output of \( M_R \) on \( x \) is \( O(\log n) \) as \( |Rx| = O(n^k) \) and we need only indices stored in binary.
Nondeterministic Log-Space

Motivation

- The intuition of a complexity class is best understood by looking at “natural” problems which are complete for this class.
- The complexity of a problem is only understood if we manage to show its completeness in a “natural” complexity class.

Theorem

REACHABILITY is NL-complete (w.r.t. log-space reductions), i.e.

- It can be decided by an NTM in space \(O(\log(n))\).
- Any problem in NL can be reduced to it in log-space.

Proof sketch of the NL-membership

Let \((V, E)\) be a graph with vertices \(V = \{1, \ldots, n\}\). Moreover, suppose that we attempt to find a path from vertex 1 to \(n\). We sketch an NTM \(N\) with input tape plus 3 worktapes:

1. On tape 2, store the current node \(i\) in binary; initially take node 1.
2. On tape 3, “guess” an integer \(j \leq n\) (= next node in the path) and check that \((i, j) \in E\).
3. If \((i, j) \in E\), then continue at step 2 with \(j\) as the new current node. Moreover, if \(j = n\), then halt with “yes”. If \((i, j) \notin E\), then halt with “no”.
4. On tape 4, maintain a counter which checks that we do not construct paths of length \(\geq n\).

Proof sketch of the NL-hardness

Let \(P\) be an arbitrary problem in NL, i.e., \(P\) is decided by a \(k\)-string nondeterministic TM \(M\) with input tape within space \(f(n) = O(\log(n))\).

Let \(x\) be an arbitrary instance of \(P\). From this, we construct the instance \(R(x) = (G, u, v)\) of REACHABILITY as follows:

The configuration graph \(G(M, x)\) of \(M\) has as its nodes all possible configurations of \(M\) and there is an edge between two nodes (configurations) \(C_1\) and \(C_2\) iff \(C_1 \xrightarrow{M} C_2\).

Our graph \(G\) contains an additional node “success” and there is an edge from any configuration \(C\) with state “yes” to the “success” node.

Finally, we set \(u = C_0 = (s, \triangleright, x, \triangleright, \triangleright, \ldots, \triangleright, \triangleright)\) (= the initial configuration of \(M\) on input \(x\)) and \(v = \text{“success”}\).

Proof sketch of the NL-hardness (continued)

Clearly, this problem reduction is correct, i.e., there exists an accepting computation of \(M\) on input \(x\) iff there exists a path in graph \(G\) from node \(u = C_0\) to node \(v = \text{“success”}\).

It remains to show that \(R(x)\) can be computed (by a deterministic TM \(N\) with input and output) in log-space.

Sketch of a log-space TM \(N\).

- \(N\) has 3 worktapes. The first two worktapes are used to store one possible configuration of \(M\) at a time.
- We are assuming that \(M\) is a \(k\)-string Turing machine with input. Hence, configurations are represented as \((q, i, w_2, u_2, \ldots, w_k, u_k)\) where \(1 \leq i \leq n + 1\) gives the cursor position on the input string. Clearly, position \(i\) as well as each string \(u_j, w_j\) fits into log-space.
Proof sketch of the NL-hardness (continued)

Operating principles of the log-space TM $N$.

- In a loop, $N$ generates all possible configurations of $M$ on its first worktape and writes each configuration to the output tape. At the end, also “success” is output.
- In a nested loop, $N$ generates all possible configurations of $M$ plus the additional node “success” on each of its first two worktapes.
- For every pair of configurations $C_1$ and $C_2$, $N$ checks (by using the third worktape) if $C_1 \overset{M}{\rightarrow} C_2$ holds. If this is the case, then $N$ writes the pair (= edge) $(C_1, C_2)$ to the output.
- $N$ also outputs all edges $(C, “success”) if C corresponds to an accepting configuration.
- Finally, the nodes $u = C_0 = (s, 0, \triangleright, \epsilon, \ldots, \triangleright, \epsilon)$ and $v = “success”$ are output.

Proper Complexity Functions

Definition

A function $f : N \rightarrow N$ is a proper complexity function if $f$ is nondecreasing and there is a $k$-string TM $M_f$ with input and output such that on any input $x$,

1. $M_f(x) = \Gamma^{f(|x|)}$ where $\Gamma$ is a quasi-blank symbol,
2. $M_f$ halts after $O(|x| + f(|x|))$ steps, and
3. $M_f$ uses $O(f(|x|))$ space besides its input.

Remark

For the definition of complexity classes $\text{TIME}(f(n))$, $\text{NTIME}(f(n))$, $\text{SPACE}(f(n))$, $\text{NSPACE}(f(n))$, we have to avoid the situation that a function $f$ cannot be computed within the time or space it allows.

Nondeterministic Space vs. Deterministic Time

Motivation

- The NL-hardness proof of REACHABILITY implicitly establishes that $\text{NL} \subseteq \text{P}$ holds.
- This result can be rephrased as follows: a nondeterministic machine $M$ working in space $f(n) = \log n$ can be simulated by a deterministic machine $N$, s.t. the time bound of $N$ is exponential w.r.t. $f(n)$.
- We want to generalize this relationship between space and time complexity to any “reasonable” function $f(n) \geq \log n$.
- The idea of “reasonable” functions is formalized by the definition of proper complexity functions.

Theorem

For any proper complexity function $f(n) \geq \log n$, we have

$$\text{NSPACE}(f(n)) \subseteq \text{TIME}(c^{f(n)})$$

Proof sketch

- We construct the configuration graph as in the NL-hardness proof of REACHABILITY.
- We get the following bound on the number of possible configurations represented as $(q, i, w_2, u_2, \ldots, w_k, u_k)$ where $1 \leq i \leq n + 1$:
  $$|K|(n + 1)(|\Sigma|^{f(n)})^{2^{(k-1)}} \leq |K|2n(|\Sigma|^{2^{(k-1)}})^{f(n)} \leq$$
  $$n c_1^{f(n)} \leq c_1^{\log n + f(n)} \leq c_1^{2f(n)} \leq c_1^{f(n)}$$
- Hence, deciding if $x$ is a positive instance of problem $\mathcal{P}$ reduces to a reachability problem for a graph with at most $c_1^{f(n)}$ nodes.
Savitch's Theorem

Theorem

\[
\text{REACHABILITY} \in \text{SPACE}(\log^2 n).
\]

Proof sketch

- Given a graph \( G \) and nodes \( x, y \) and \( i \geq 0 \), define \( \text{PATH}(x, y, i) \) as the assertion "there is a path from \( x \) to \( y \) of length at most \( 2^i \)."
- If \( G \) has \( n \) nodes then any cycle-free path has length \( \leq n \) and we can solve reachability in \( G \) if we can compute whether \( \text{PATH}(x, y, \lceil \log n \rceil) \) holds for any given nodes \( x, y \) of \( G \).
- This can be done using middle-first search.

Proof sketch (continued)

- Proof that \( \text{path}(x, y, i) \) correctly determines \( \text{PATH}(x, y, i) \) (by induction on \( i \)):
  - \( i = 0 \). Clearly, \( \text{path}(x, y, 0) \) correctly determines \( \text{PATH}(x, y, 0) \).
  - \( i > 0 \). \( \text{path}(x, y, i) \) returns "yes" iff there is a node \( z \) with both \( \text{path}(x, z, i - 1) \) and \( \text{path}(z, y, i - 1) \) holding.

By the induction hypothesis, \( \text{path}(x, z, i - 1) \) and \( \text{path}(z, y, i - 1) \) return "yes" iff there are paths from \( x \) to \( z \) and from \( z \) to \( y \) both at most \( 2^{i-1} \) long.

This is the case iff there is a path from \( x \) to \( y \) at most \( 2^i \) long.

Proof sketch (continued)

- Idea of the middle-first search. Guess a midpoint of the alleged path and recursively check for the existence of a half-length path from the start to the midpoint and from the midpoint to the finish.
- Implementation of the middle-first search.

\[
\text{function } \text{path}(x, y, i) /* middle-first search */
\]

- if \( i = 0 \) then
  - if \( x = y \) or there is an edge \((x, y)\) in \( G \) then return "yes";
  - else for all nodes \( z \) do
    - if \( \text{path}(x, z, i - 1) \) and \( \text{path}(z, y, i - 1) \) then return "yes";
    - return "no".

- The algorithm is started with \( \text{path}(x, y, \lceil \log n \rceil) \).
- The \( O(\log^2 n) \) space bound is achieved by handling recursion using a stack containing a triple \((x, y, i)\) for each active recursive call.
  - For each node \( z \) put \((x, z, i - 1)\) onto the stack and call \( \text{path}(x, z, i - 1) \).
  - If this fails, erase \((x, z, i - 1)\) and put \((x, z', i - 1)\) for the next \( z' \).
    - Otherwise erase \((x, z, i - 1)\) and continue with \((z, y, i - 1)\).
- As there are at most \( \log n \) recursive calls active with each taking at most \( 3 \log n \) space, the \( O(\log^2 n) \) space bound is achieved.
Nondeterministic Space vs. Deterministic Space

**Corollary**
*For any proper complexity function \( f(n) \geq \log n \), we have*
\[
\text{NSPACE}(f(n)) \subseteq \text{SPACE}((f(n))^2)
\]

**Proof sketch**
- To simulate an \( f(n) \)-space bounded NTM \( M \) on input \( x \), run the previous algorithm on the configuration graph \( G(M, x) \).
- The edges of the graph \( G(M, x) \) are determined on the fly by examining the input \( x \).
- The configuration graph has at most \( c^f(n) \) nodes.
- By Savitch’s Theorem, the algorithm needs at most \((\log c^f(n))^2 = f(n)^2 \log^2 c = O(f(n)^2)\) space.

Basic Complexity Classes Revisited

**Theorem**
\( L \subseteq \text{NL} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{NEXPTIME} \subseteq \text{EXPSPACE} \)

**Proof**
In the “Formale Methoden” lecture, we showed all inclusions except for \( \text{NL} \subseteq \text{P} \) and \( \text{PSPACE} \subseteq \text{EXPTIME} \).
These inclusions follow from the more general relationship
\[
\text{NSPACE}(f(n)) \subseteq \text{TIME}(c^{f(n)})
\]

**Remark**
It is now clear, why we have not mentioned \( \text{NPSPACE} \) and \( \text{NEXPSPACE} \).
Indeed, \( \text{PSPACE} = \text{NPSPACE} \) and \( \text{EXPSPACE} = \text{NEXPSPACE} \) hold.

Immerman-Szelepscényi Theorem

**Theorem**
*Given a graph \( G \) and a node \( u \), the number of nodes reachable from \( u \) in \( G \) can be computed by an NTM within logarithmic space.*

More formally, given a graph \( G \), a node \( u \), and an integer \( m \), deciding if the number of nodes reachable from \( u \) is \( m \) can be done in \( \text{NL} \).

**Theorem**
\( \text{REACHABILITY} \) is in \( \text{co-NL} \). Hence, \( \text{NL} = \text{co-NL} \).

**Theorem**
*If \( f(n) \geq \log n \) is a proper complexity function, then \( \text{NSPACE}(f(n)) = \text{co-NSPACE}(f(n)) \).*
### co-NL-Membership of REACHABILITY

**Proof Idea**

What shall a witness look like that there is no path from \( u \) to \( v \)?

**Idea.** Let \( S(k) \) denote the set of nodes reachable from \( u \) in \( k \) steps (with \( 0 \leq k \leq n - 1 \)) and suppose that we know \( m = |S(n - 1)| \).

Then a witness \( y \) for not-reachability consists of \( m \) paths from \( u \) to (pairwise distinct) vertices \( v_1, \ldots, v_m \in V \), s.t. \( v_i \neq v_j \) for all \( i \).

---

**Logspace verification of such witnesses**

```plaintext
for \( j := 1, 2, \ldots, n \) do {
    guess flag; /* meaning: flag = true if \( v_j \in S(n - 1) \) */
    if flag then {
        decrement \( m \);
        guess a path \( \pi \) from \( u \) to \( v_j \) of length \( \leq n - 1 \);
        check that \( v \neq v_j \); }
    check that \( m = 0 \);
}
```

---

**Idea of the Algorithm (continued)**

- Clearly, we can check \( v_j \in S(k) \) in NL. It remains to show that we can also check \( v_j \notin S(k) \) in NL in the inner loop of our algorithm.
- When checking \( v_j \notin S(k) \), we already know \( m = |S(k - 1)| \).
- A witness \( y \) for \( v_j \notin S(k) \) consists of \( m \) paths from \( u \) to vertices \( v_1, \ldots, v_m \in V \), s.t. \( v_i \neq v_j \) and \( (v_i, v_j) \notin E \).
- Below we give a verification of witness \( y \) that requires only logspace.

```plaintext
for \( v := 1, 2, \ldots, n \) do {
    guess flag; /* meaning: flag = true if \( v \in S(k - 1) \) */
    if flag then {
        decrement \( m \);
        guess a path \( \pi \) from \( u \) to \( v \) of length \( \leq k - 1 \);
        check that \( v \neq v_j \) and \( (v, v_j) \notin E \); }
    check that \( m = 0 \);
```

---

**Learning Objectives**

- Computational power of log-space
- Reductions in this lecture: log-space, many-one
- Composability of reductions
- NL-completeness of REACHABILITY (configuration graph)
- Savitch's Theorem (middle-first search)
- Basic relationships (inclusions) between complexity classes
- Immerman-Szelepcscényi Theorem