Theory of Languages and Automata

Chapter 9- Intractability

Sharif University of Technology
Space Hierarchy Theorem

**Definition 9.1**

A function $f: \mathbb{N} \rightarrow \mathbb{N}$, where $f(n)$ is at least $O(\log n)$, is called *space constructible* if the function that maps the string $1^n$ to the binary representation of $f(n)$ is computable in space $O(f(n))$.\(^1\)
Example 9.2

All commonly occurring functions that are at least $O(\log n)$ are space constructible, including the functions $\log_2 n$, $n \log_2 n$, and $n^2$.

For example, $n^2$ is space constructible because a machine may take its input $1^n$, obtain $n$ in binary by counting the number of 1s, and output $n^2$ by using any standard method for multiplying $n$ by itself. The total space used is $O(n)$ which is certainly $O(n^2)$.
THEOREM 9.3  

Space hierarchy theorem  For any space constructible function \( f: \mathcal{N} \rightarrow \mathcal{N} \), a language \( A \) exists that is decidable in \( O(f(n)) \) space but not in \( o(f(n)) \) space.
Proof Idea

**Proof Idea** We must demonstrate a language $A$ that has two properties. The first says that $A$ is decidable in $O(f(n))$ space. The second says that $A$ isn’t decidable in $o(f(n))$ space.

We describe $A$ by giving an algorithm $D$ that decides it. Algorithm $D$ runs in $O(f(n))$ space, thereby ensuring the first property. Furthermore, $D$ guarantees that $A$ is different from any language that is decidable in $o(f(n))$ space, thereby ensuring the second property. Language $A$ is different from languages we have discussed previously in that it lacks a nonalgorithmic definition. Therefore we cannot offer a simple mental picture of $A$.

In order to ensure that $A$ not be decidable in $o(f(n))$ space, we design $D$ to implement the diagonalization method that we used to prove the unsolvability of the halting problem $A_{TM}$ in Theorem 4.11 on page 174. If $M$ is a $TM$ that decides a language in $o(f(n))$ space, $D$ guarantees that $A$ differs from $M$’s language in at least one place. Which place? The place corresponding to a description of $M$ itself.
The following $O(f(n))$ space algorithm $D$ decides a language $A$ that is not decidable in $o(f(n))$ space.

$D =$ “On input $w$:

1. Let $n$ be the length of $w$.
2. Compute $f(n)$ using space constructibility, and mark off this much tape. If later stages ever attempt to use more, reject.
3. If $w$ is not of the form $\langle M \rangle 10^*$ for some TM $M$, reject.
4. Simulate $M$ on $w$ while counting the number of steps used in the simulation. If the count ever exceeds $2^{f(n)}$, reject.
5. If $M$ accepts, reject. If $M$ rejects, accept.”
**Corollary 9.4**

For any two functions \( f_1, f_2 : \mathcal{N} \rightarrow \mathcal{N} \), where \( f_1(n) \) is \( o(f_2(n)) \) and \( f_2 \) is space constructible, \( \text{SPACE}(f_1(n)) \subsetneq \text{SPACE}(f_2(n)) \).

---

**Corollary 9.5**

For any two real numbers \( 0 \leq \epsilon_1 < \epsilon_2 \),

\[
\text{SPACE}(n^{\epsilon_1}) \subsetneq \text{SPACE}(n^{\epsilon_2}).
\]
Main Corollaries

**COROLLARY 9.6**

\[ \text{NL} \not\subseteq \text{PSPACE}. \]

**PROOF** Savitch’s theorem shows that \( \text{NL} \subseteq \text{SPACE}(\log^2 n) \), and the space hierarchy theorem shows that \( \text{SPACE}(\log^2 n) \not\subseteq \text{SPACE}(n) \). Hence the corollary follows. As we observed on page 326, this separation allows us to conclude that \( \text{TQBF} \not\in \text{NL} \) because \( \text{TQBF} \) is PSPACE-complete with respect to log space reducibility.
Main Corollaries

Now we establish the main objective of this chapter: proving the existence of problems that are decidable in principle but not in practice—that is, problems that are decidable but intractable. Each of the classes $\text{SPACE}(n^k)$ is contained within the class $\text{SPACE}(n^{\log n})$, which in turn is strictly contained within the class $\text{SPACE}(2^n)$. Therefore we obtain the following additional corollary separating $\text{PSPACE}$ from $\text{EXPSPACE} = \bigcup_k \text{SPACE}(2^{n^k})$.

**Corollary 9.7**

$\text{PSPACE} \subsetneq \text{EXPSPACE}$. 
Time Hierarchy Theorem

**Definition 9.8**

A function \( t : \mathbb{N} \rightarrow \mathbb{N} \), where \( t(n) \) is at least \( O(n \log n) \), is called *time constructible* if the function that maps the string \( 1^n \) to the binary representation of \( t(n) \) is computable in time \( O(t(n)) \).

**Example 9.9**

All commonly occurring functions that are at least \( n \log n \) are time constructible, including the functions \( n \log n, n\sqrt{n}, n^2, \) and \( 2^n \).

For example, to show that \( n\sqrt{n} \) is time constructible, we first design a TM to count the number of 1s in binary. To do so the TM moves a binary counter along the tape, incrementing it by 1 for every input position, until it reaches the end of the input. This part uses \( O(n \log n) \) steps because \( O(\log n) \) steps are used for each of the \( n \) input positions. Then, we compute \( \lfloor n\sqrt{n} \rfloor \) in binary from the binary representation of \( n \). Any reasonable method of doing so will work in \( O(n \log n) \) time because the length of the numbers involved is \( O(\log n) \).
THEOREM 9.10

Time hierarchy theorem  For any time constructible function $t: \mathbb{N} \rightarrow \mathbb{N}$, a language $A$ exists that is decidable in $O(t(n))$ time but not decidable in time $o(t(n)/\log t(n))$.

PROOF IDEA  This proof is similar to the proof of Theorem 9.3. We construct a TM $D$ that decides a language $A$ in time $O(t(n))$, whereby $A$ cannot be decided in $o(t(n)/\log t(n))$ time. Here, $D$ takes an input $w$ of the form $\langle M \rangle 10^*$ and simulates $M$ on input $w$, making sure not to use more than $t(n)$ time. If $M$ halts within that much time, $D$ gives the opposite output.
Proof:

The following $O(t(n))$ time algorithm $D$ decides a language $A$ that is not decidable in $o(t(n)/ \log t(n))$ time.

$D =$ “On input $w$:
1. Let $n$ be the length of $w$.
2. Compute $t(n)$ using time constructibility, and store the value $[t(n)/ \log t(n)]$ in a binary counter. Decrement this counter before each step used to carry out stages 3, 4, and 5. If the counter ever hits 0, reject.
3. If $w$ is not of the form $\langle M \rangle 10^*$ for some TM $M$, reject.
4. Simulate $M$ on $w$.
5. If $M$ accepts, then reject. If $M$ rejects, then accept.”

We examine each of the stages of this algorithm to determine the running time. Obviously, stages 1, 2 and 3 can be performed within $O(t(n))$ time.

In stage 4, every time $D$ simulates one step of $M$, it takes $M$’s current state together with the tape symbol under $M$’s tape head and looks up $M$’s next action in its transition function so that it can update $M$’s tape appropriately. All three of these objects (state, tape symbol, and transition function) are stored on $D$’s tape somewhere. If they are stored far from each other, $D$ will need many steps to gather this information each time it simulates one of $M$’s steps. Instead, $D$ always keeps this information close together.
Corollaries

**Corollary 9.11**

For any two functions \( t_1, t_2 : \mathcal{N} \rightarrow \mathcal{N} \), where \( t_1(n) \) is \( o(t_2(n)/\log t_2(n)) \) and \( t_2 \) is time constructible, \( \text{TIME}(t_1(n)) \subsetneq \text{TIME}(t_2(n)) \).

**Corollary 9.12**

For any two real numbers \( 1 \leq \epsilon_1 < \epsilon_2 \),

\[
\text{TIME}(n^{\epsilon_1}) \subsetneq \text{TIME}(n^{\epsilon_2}).
\]
Main Corollary

COROLLARY 9.13

P \nsubseteq \text{EXPTIME}. 
Exponential Space Completeness

We show that, by allowing regular expressions with more operations than the usual regular operations, the complexity of analyzing the expressions may grow dramatically. Let ↑ be the \textit{exponentiation operation}. If \( R \) is a regular expression and \( k \) is a nonnegative integer, writing \( R \uparrow k \) is equivalent to the concatenation of \( R \) with itself \( k \) times. We also write \( R^k \) as shorthand for \( R \uparrow k \). In other words,

\[
R^k = R \uparrow k = \underbrace{R \circ R \circ \cdots \circ R}_{k}
\]
EXPSPACE-Completeness

\[ EQ_{\text{REX}}^\uparrow = \{ \langle Q, R \rangle | Q \text{ and } R \text{ are equivalent regular expressions with exponentiation} \} \]

To show that \( EQ_{\text{REX}}^\uparrow \) is intractable we demonstrate that it is complete for the class EXPSPACE. Any EXPSPACE-complete problem cannot be in PSPACE, much less in P. Otherwise EXPSPACE would equal PSPACE, contradicting Corollary 9.7.
EXPSPACE Completeness

**Definition 9.14**

A language $B$ is **EXPSPACE-complete** if

1. $B \in \text{EXPSPACE}$, and
2. every $A$ in EXPSPACE is polynomial time reducible to $B$. 
Theorem

**Theorem 9.15**

$EQ_{REX^\uparrow}$ is EXPSPACE-complete.

**Proof Idea** In measuring the complexity of deciding $EQ_{REX^\uparrow}$, we assume that all exponents are written as binary integers. The length of an expression is the total number of symbols that it contains.

We sketch an EXPSPACE algorithm for $EQ_{REX^\uparrow}$. To test whether two expressions with exponentiation are equivalent, we first use repetition to eliminate exponentiation, then convert the resulting expressions to NFAs. Finally, we use an NFA equivalence testing procedure similar to the one used for deciding the complement of $ALL_{NFA}$ in Example 8.4.
Proof:

First we present a nondeterministic algorithm for testing whether two NFAs are inequivalent.

\[ N = \text{“On input } \langle N_1, N_2 \rangle, \text{ where } N_1 \text{ and } N_2 \text{ are NFAs:} \]

1. Place a marker on each of the start states of \( N_1 \) and \( N_2 \).
2. Repeat \( 2^{q_1 + q_2} \) times, where \( q_1 \) and \( q_2 \) are the numbers of states in \( N_1 \) and \( N_2 \):
3. Nondeterministically select an input symbol and change the positions of the markers on the states of \( N_1 \) and \( N_2 \) to simulate reading that symbol.
4. If at any point, a marker was placed on an accept state of one of the finite automata and not on any accept state of the other finite automaton, accept. Otherwise, reject.”
Algorithm $N$ runs in nondeterministic linear space and thus, by applying Savitch’s theorem, we obtain a deterministic $O(n^2)$ space algorithm for this problem. Next we use the deterministic form of this algorithm to design the following algorithm $E$ that decides $EQ_{\text{REX}^\uparrow}$.

$E =$ “On input $\langle R_1, R_2 \rangle$ where $R_1$ and $R_2$ are regular expressions with exponentiation:

1. Convert $R_1$ and $R_2$ to equivalent regular expressions $B_1$ and $B_2$ that use repetition instead of exponentiation.
2. Convert $B_1$ and $B_2$ to equivalent NFA$s N_1$ and $N_2$, using the conversion procedure given in the proof of Lemma 1.55.
3. Use the deterministic version of algorithm $N$ to determine whether $N_1$ and $N_2$ are equivalent.”

Algorithm $E$ obviously is correct. To analyze its space complexity we observe that using repetition to replace exponentiation may increase the length of an expression by a factor of $2^l$, where $l$ is the sum of the lengths of the exponents.