CS:4330 Theory of Computation

Spring 2018

Computability Theory

Space Complexity

Haniel Barbosa

The University of Iowa

Readings for this lecture

Chapter 8 of [Sipser 1996], 3rd edition. Sections 8.1, 8.2, and 8.3.

Space Complexity

- We now consider the complexity of computational problems in terms of the amount of space, or memory, they require
- Time and space are two of the most important considerations when we seek practical solutions to most problems
- Space complexity shares many of the features of time complexity
- It serves a further way of classifying problems according to their computational difficulty

Space Complexity

Definition

Let M be a deterministic Turing machine, DTM, that halts on all inputs. The space complexity of M is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of tape cells that M scans on any input of length n.

Definition

If M is a nondeterministic Turing machine, NTM, wherein all branches of its computation halt on all inputs, we define the space complexity of M, $f(n)$, to be the maximum number of tape cells that M scans on any branch of its computation for any input of length n.

Estimation of space complexity

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. The space complexity classes, $\mathrm{SPACE}(f(n))$ and $\mathrm{NSPACE}(f(n))$, are defined by:

- $\mathrm{SPACE}(f(n)) = \{L \mid L$ is a language decided by an $\mathcal{O}(f(n))$ space DTM$\}$

- $\mathrm{NSPACE}(f(n)) = \{L \mid L$ is a language decided by an $\mathcal{O}(f(n))$ space NTM$\}$

Example

SAT can be solved with the linear space algorithm M_1:
$M_1 =$"On input $\langle \varphi \rangle$, where φ is a Boolean formula:

1. For each truth assignment to the variables $x_1, \ldots, x_n$ of φ:
   (a) Evaluate φ on that truth assignment;
   (b) If φ evaluates to 1, *accept*
2. If φ never evaluates to 1, *reject*."

Another Example

Testing whether a DFA accepts all strings,

$$ALL_{\text{DFA}} = \{\langle A \rangle \mid A \text{ is a DFA and } \mathscr{L}(A) = \Sigma^*\}$$
$$\overline{ALL_{\text{DFA}}} = \{\langle A \rangle \mid A \text{ is a DFA and } \mathscr{L}(A) \neq \Sigma^*\}$$

We show that $\overline{ALL_{\text{DFA}}} \in \text{SPACE}(n)$

Proof idea

Construct D, a deterministic linear space algorithm that decides $\overline{ALL_{\text{DFA}}}$ by enumerating all strings of length n. If A rejects one of them, D stops with "accept". If A accepts all, D will "reject."

Yet Another Example

Testing whether an NFA accepts all strings,

$$ALL_{\text{NFA}} = \{\langle A \rangle \mid A \text{ is an NFA and } \mathscr{L}(A) = \Sigma^*\}$$
$$\overline{ALL_{\text{NFA}}} = \{\langle A \rangle \mid A \text{ is an NFA and } \mathscr{L}(A) \neq \Sigma^*\}$$

We show that $\overline{ALL_{\text{NFA}}} \in \text{NSPACE}(n)$

Proof idea

Construct N, a nondeterministic linear space algorithm that decides $\overline{ALL_{\text{NFA}}}$ by guessing a string that is rejected by NFA A and uses a linear space to keep track of which states the NFA could be in at a particular time. Note, this language is not know to NP or coNP.

Construction

$N =$"On input $\langle A \rangle$, where A is an NFA:

1. Place a marker on the start state of A.
2. Repeat 2^q times, where q is the number of states of A:
   (a) Nondeterministically select an input symbol and change the positions of all the markers on A's states to simulate the reading of that symbol
   (b) If all the marked states are non-final, *accept*
3. *reject*."

SPACE vs NSPACE

Recall $\mathrm{NTIME}(f(n)) \subseteq \mathrm{TIME}(2^{\mathcal{O}(f(n))})$

Theorem (Savitch's Theorem)

For any function $f : \mathbb{N} \to \mathbb{N}$, *where* $f(n) \geq n$, $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n))$

Proof idea

If NTM N uses $f(n)$ space, we need to construct a DTM M which simulates N and uses $f^2(n)$ space.
M calls the function $CANYIELD(c_1, c_2, t)$, which can be computed by a TM, that tests whether a NTM N can go from the configuration c_1 of N to the configuration c_2 of N in at most t steps.

$CANYIELD(c_1, c_2, t)$

$CANYIELD(c_1, c_2, t)$ is a TM that tests whether a NTM N can go from configuration c_1 of N to the configuration c_2 of N in at most t steps

$CANYIELD(c_1, c_2, t) =$"On input $\langle c_1, c_2, t \rangle$:

1. If $t = 1$ test whether $c_1 = c_2$ or whether c_1 yields c_2 in one step according to the transition rules of N; *accept* if either test succeeds, *reject* if both fail
2. If $t > 1$ then for each configuration x of N using space $f(n)$:
   (a) Run $CANYIELD(c_1, x, \lceil t/2 \rceil)$
   (b) Run $CANYIELD(x, c_2, \lfloor t/2 \rfloor)$
   (c) If both $2a$ and $2b$ accept, then *accept*
3. If haven't yet accept at $2c$, *reject*"

DTM M simulating NMT N

- Suppose N is modified that once it enters an accepting state, it erases everything on the tape and moves the tape head to the leftmost position. Let such a final configuration be c_a.

- Let d be a constant such that N has no more than $2^{df(n)}$ configurations.

- Then the DTM M is:
  $M =$"On input w:
  1. Let c_s be $q_0 w$, where q_0 is the start state of N
  2. Output the result of $CANYIELD(c_s, c_a, 2^{df(n)})$."

Analyzing M

- ▷ Whenever $CANYIELD$ invokes itself recursively it stores the current stage number and the values c_1, c_2, t on the stack
- ▷ Each level of recursion uses $\mathcal{O}(f(n))$ additional space
- ▷ Each level of recursion divides the size t in half. Since initially $t = 2^{df(n)}$, the depth of the recursion is $\mathcal{O}(\log 2^{df(n)}) = \mathcal{O}(f(n))$
- ▷ Hence, the total space used is $\mathcal{O}(f^2(n))$

The class PSPACE

PSPACE is the class of languages that are decidable in polynomial space on a DTM, i.e.

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}(n^k)$$

$$\mathrm{NPSPACE} = \bigcup_k \mathrm{NSPACE}(n^k)$$

We define $\mathrm{EXPTIME} = \bigcup_k \mathrm{TIME}(2^{n^k})$

Lemma

$\mathrm{SPACE}(f(n)) \subseteq \mathrm{TIME}(2^{\mathcal{O}(f(n))})$

The class PSPACE

PSPACE is the class of languages that are decidable in polynomial space on a DTM, i.e.

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}(n^k)$$
$$\mathrm{NPSPACE} = \bigcup_k \mathrm{NSPACE}(n^k)$$

We define $\mathrm{EXPTIME} = \bigcup_k \mathrm{TIME}(2^{n^k})$

Lemma

$\mathrm{SPACE}(f(n)) \subseteq \mathrm{TIME}(2^{\mathcal{O}(f(n))})$

What we know so far

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE} \subseteq \mathrm{EXPTIME}$$

PSPACE-Completeness

Definition

If every $A \in \mathrm{PSPACE}$ is polynomial time reducible to a language B, then B is *PSPACE-hard*.

Definition

A language B is *PSPACE-complete* if it satisfies two conditions:

1. $B \in \mathrm{PSPACE}$
2. B is *PSPACE-hard*

A PSPACE-complete language

Definition

A formula φ is a *fully quantified Boolean formula* if

$$\varphi = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n . \psi$$

where ψ is a Boolean formula in CNF, $x_1, \ldots, x_n$ are the Boolean variables in ψ, and $Q_i \in \{\forall, \exists\}$, $1 \leq i \leq n$. φ is said to be in *prenex normal form*.

Consider the problem

$$TQBF = \{\langle \varphi \rangle \mid \varphi \text{ is a true fully quantified Boolean formula}\}$$

Theorem

TBQF is PSPACE-complete.

$TBQF \in$ PSPACE

The following polynomial space algorithm decides *TQBF*:

$T =$"On input $\langle \varphi \rangle$, in which φ is a fully quantified Boolean formula:

1. If φ contains no quantifiers, then it is an extension with only constants. So, evaluate φ and *accept* if it is true; otherwise, *reject*
2. If $\varphi = \exists x. \psi$, recursively call T on ψ, first with 0 substituted for x and then with 1 substituted for x. If either result is "accept", *accept*, otherwise *reject*
3. If $\varphi = \forall x. \psi$, recursively call T on ψ, first with 0 substituted for x and then with 1 substituted for x. If both results are "accept", *accept*, otherwise *reject*"

$TBQF$ is *PSPACE-hard*

We use T and the construction of $CANYIELD$ to prove that any PSPACE problem reduces to $TQBF$ in polynomial time.

Remember the polynomial reduction used to prove that SAT is *NP-hard*:

- ▷ On input w of an NTM N that recognizes language A in $\mathcal{O}(n^k)$, such that $n = |w|, f$ produces φ_w in $\mathcal{O}(n^k)$
- ▷ *Variables* of φ_w: Let $N = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ and $C = Q \cup \Gamma \cup \{\#\}$. For each $1 \leq i,j \leq n^k$ and $s \in C$ we have a variable $x_{i,j,s}$
- ▷ *Cells*: each of the $(n^k)^2$ entries of a tableau is called a *cell*. The cell in row i and column j is called $cell[i,j]$ and contains a symbol from C.
- ▷ The contents of cells are represented with the variables of φ
- ▷ For every state $s \in C$, $x_{i,j,s} = 1$ if $cell[i,j] = s$
- ▷ Formula $\varphi_w = \varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{move} \wedge \varphi_{accept}$ corresponds to $\text{Tableau}(N,w)$

Remember: $\mathrm{Tableau}(N, w)$ **must be well formed**

We first ensure that the tableau is well formed, i.e. it contains exactly one symbol per cell. Thus φ_w must guarantee that exactly one variable is true for each cell:

1. at least one variable that is associated with a cell is true, represented by

$$\bigvee_{s \in C} x_{i,j,s}$$

2. no more than one variable is true for each cell, i.e. for each pair of variables at least one is false, which is represented by

$$\bigwedge_{s,t \in C, s \neq t} (\neg x_{i,j,s} \vee \neg x_{i,j,t})$$

A satisfying assignment which makes true one variable for each cell is specified by

$$\varphi_{cell} = \bigwedge_{1 \leq i,j \leq n^k} \left[\bigvee_{s \in C} x_{i,j,s} \wedge \left(\bigwedge_{s,t \in C, s \neq t} (\neg x_{i,j,s} \vee \neg x_{i,j,t}) \right) \right]$$

Remember: $\mathrm{Tableau}(N, w)$ must be an accepting tableau

The formulas $\varphi_{start}, \varphi_{move}, \varphi_{accept}$ guarantee that the symbols in the tableau actually correspond to an accepting tableau

- ▷ The first row of the tableau must be the starting configuration of N on w:

$$\varphi_{start} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,w_1} \wedge \cdots \wedge x_{1,n+2,w_n} \wedge x_{1,n+3,\sqcup} \wedge \cdots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}$$

- ▷ An accepting configuration must occur in the tableau, i.e. q_a must be in one of the cells:

$$\varphi_{accept} = \bigvee_{1 \leq i,j \leq n^k} x_{i,j,q_a}$$

- ▷ Each row of the tableau must correspond to a configuration that legally follows the preceding row's configuration according to N's transition rules. This is guaranteed by φ_{move}

Remember: Legal windows

A 2×3 window of cells is legal if that window does not violate the actions specified by N's transition function. In other words, a window is legal if it might appear when one configuration correctly follows another.

Consider the transitions:

$$\delta(q_1, \mathrm{a}) = \{(q_1, \mathrm{b}, R)\}, \delta(q_1, \mathrm{b}) = \{(q_2, \mathrm{c}, L), (q_2, \mathrm{a}, R)\}$$

Examples of legal windows for this machine are:

(a)

a	q_1	b
q_2	a	c

(b)

a	q_1	b
a	a	q_2

(c)

a	a	q_1
a	a	b

(d)

#	b	a
#	b	a

(e)

a	b	a
a	b	q_2

(f)

b	b	b
c	b	b

TBQF is *PSPACE-hard*

Let A be a language decided by TM M in space n^k for some constant k. We give a polynomial time reduction from A to *TQBF*:

The reduction maps machine M and string w to a quantified Boolean formula φ that is true iff M accepts w:

- ▷ The formula is $\varphi_{c_1,c_2,t}$, where c_1 and c_2 are variables representing two configurations, and $t > 0$
- ▷ If we assign to c_1 and c_2 actual configurations, $\varphi_{c_1,c_2,t}$ is true iff M can go from c_1 to c_2 in at most t steps
- ▷ Consider $\varphi_{c_s,c_a,2^{df(n)}}$, where d is a constant chosen so that M has no more than $2^{df(n)}$ configurations. Assume w.l.o.g. that t is a power of 2.

Note

- $\varphi_{c_1,c_2,t}$ encodes the contents of tape cells as in the Cook-Levin theorem. Each configuration has n^k cells and so it is encoded by $\mathcal{O}(n^k)$ variables
- For $t = 1$, the formula $\varphi_{c_1,c_2,t}$ says that $c_1 = c_2$ or c_2 follows from c_1 in a single step of M
- If $t > 1$, construct $\varphi_{c_1,c_2,t}$ recursively:

$$\varphi_{c_1,c_2,t} = \exists m_1 [\varphi_{c_1,m,t/2} \wedge \varphi_{m_1,c_2,t/2}]$$

  where m_1 is a configuration of M. The notation $\exists m_1$ is shorthand for $\exists x_1, \ldots, x_l$, such that $l = \mathcal{O}(n^k)$, $x_1, \ldots, x_l$ are variable encoding m_1
- To reduce the size of the formula we use both quantifiers, $\forall$ and $\exists$:

$$\varphi_{c_1,c_2,t} = \exists m_1 \forall (c_3, c_4) \in \{(c_1, m_1), (m_1, c_2)\} [\varphi_{c_3,c_3,t/2}]$$

  where $\forall x \in \{y, z, \ldots\}. \varphi(x)$ denotes $(\varphi(y) \wedge P(z) \wedge \ldots)$

Complexity of the reduction

- ▷ We need to show that the size of $\varphi_{c_s,c_a,2^{df(n)}}$ is polynomial on n
- ▷ Each level of the recursion adds a portion of the formula that is linear in the size of the configurations is thus of size $\mathcal{O}(f(n))$
- ▷ The number of levels of the recursion is $\log 2^{df(n)}$, or $\mathcal{O}(f(n))$.
- ▷ Therefore the size of the resulting formula is $\mathcal{O}(f^2(n))$

This concludes our proof that $TQBF$ is *PSPACE-hard*. Since it is also *PSPACE*, we have proved that $TQBF$ is *PSPACE-complete*.