UMB CS622

PSPACE Completeness

Monday, November 29, 2021
Announcements

• HW 9 extended
  • Due Tues 11/30 11:59pm EST

• HW 10 released
  • Due Tues 12/7 11:59pm EST

• HW 11 will be last assignment
  • Due Tues 12/14 11:59pm EST
**Flashback: Dynamic Programming Example**

- **Chomsky Grammar** $G$:
  - $S \rightarrow AB \mid BC$
  - $A \rightarrow BA \mid a$
  - $B \rightarrow CC \mid b$
  - $C \rightarrow AB \mid a$

- **Example string**: *baaba*

- Store every **partial string** and their generating variables in a **table**

<table>
<thead>
<tr>
<th>Substring start char</th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>vars for “b”</td>
<td>vars for “ba”</td>
<td>vars for “baa”</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td>vars for “a”</td>
<td>vars for “aa”</td>
<td>vars for “aab”</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We are gaining time ...

... by spending more space!
Space Complexity, Formally

**Definition**

Let $M$ be a deterministic Turing machine 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 $n$. If the space complexity of $M$ is $f(n)$, we also say that $M$ runs in space $f(n)$.

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

TMs have a space complexity

decider
Space Complexity Classes

**Definition**

Let $f: \mathbb{N} \rightarrow \mathbb{R}^+$ be a function. The *space complexity classes*, $\text{SPACE}(f(n))$ and $\text{NSPACE}(f(n))$, are defined as follows.

- $\text{SPACE}(f(n)) = \{L | L \text{ is a language decided by an } O(f(n)) \text{ space deterministic Turing machine}\}$.
- $\text{NSPACE}(f(n)) = \{L | L \text{ is a language decided by an } O(f(n)) \text{ space nondeterministic Turing machine}\}$.

**Compare:**

Let $t: \mathbb{N} \rightarrow \mathbb{R}^+$ be a function. Define the *time complexity class*, $\text{TIME}(t(n))$, to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

$\text{NTIME}(t(n)) = \{L | L \text{ is a language decided by an } O(t(n)) \text{ time nondeterministic Turing machine}\}$. 
Example: \textit{SAT} Space Usage

\[ SAT = \{ \langle \phi \rangle | \phi \text{ is a satisfiable Boolean formula} \} \]

2\(^{O(m)}\) exponential time machine

\[
M_1 = \text{"On input } \langle \phi \rangle, \text{ where } \phi \text{ is a Boolean formula:}
\]
1. For each truth assignment to the variables \(x_1, \ldots, x_m\) of \(\phi\):
2. Evaluate \(\phi\) on that truth assignment.
3. If \(\phi\) ever evaluated to 1, \textit{accept}; if not, \textit{reject}.

Each loop iteration requires \(O(m)\) space

But the space is re-used on each loop! (nothing is stored from the prev loop)

So this machine runs in \(O(m)\) space complexity!

Space is "more powerful" than time.

\(SAT\) is in \(O(m)\) space complexity class!
Example: Nondeterministic Space Usage

\[ \text{ALL}_{\text{NFA}} = \{ \langle A \rangle \mid A \text{ is an NFA and } L(A) = \Sigma^* \} \]

Nondeterministic decider for \( \text{ALL}_{\text{NFA}} \) (accepts NFAs that reject something)

\[ N = \text{“On input } \langle M \rangle, \text{ where } M \text{ is an NFA:} \]

1. Place a marker on the start state of the NFA.
2. Repeat \( 2^q \) times, where \( q \) is the number of states of \( M \):
   - Nondeterministically select an input symbol and change the positions of the markers on \( M \)'s states to simulate reading that symbol.
3. Accept if stages 2 and 3 reveal some string that \( M \) rejects; that is, if at some point none of the markers lie on accept states of \( M \). Otherwise, reject.”

\[ q \text{ states} = 2^q \text{ possible combinations (so exponential time)} \]

Additionally, need a counter to count to \( 2^q \); this requires \( \log (2^q) = q \) extra space

So the whole machine runs in (nondeterministic) linear \( O(q) \) space!

Machine tracks “current” state(s) of NFA

But each loop uses only \( O(q) \) space!
Facts About Time vs Space (for Deciders)

**TIME → SPACE**

- If a decider runs in time \( t(n) \), then its maximum space usage is ...
- \( \ldots t(n) \)
- \( \ldots \) because it can add at most 1 tape cell per step

**SPACE → TIME**

- If a decider runs in space \( f(n) \), then its maximum time usage is ...
- \( \ldots (|\Gamma| + |Q|)^{f(n)} = 2^{df(n)} \)
- \( \ldots \) because that’s the number of possible configurations
- (and a decider cannot repeat a configuration)

What about deterministic vs non-deterministic?
Flashback: Deterministic vs Non-Det. Time

- If a non-deterministic TM runs in: $t(n)$ time
- Then an equivalent deterministic TM runs in: $2^{O(t(n))}$
  - Exponentially slower

What about space?
Deterministic vs Non-Det. Space

**THEOREM**

**Savitch’s theorem**  For any function $f: \mathbb{N} \rightarrow \mathbb{R}^+$, where $f(n) \geq n$, \n\nspace

\[ \text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n)). \]

• If a non-deterministic TM runs in: $f(n)$ space

• Then an equivalent deterministic TM runs in: $f^2(n)$ space
  • Exponentially Only Quadratically slower!
Flashback: Nondet $\Rightarrow$ Deterministic TM: Time

$t(n)$ time $\Rightarrow 2^{O(t(n))}$ time

- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Deterministically check every tree path, in breadth-first order
    - 1
    - 1-1
    - 1-2
    - 1-1-1

Nondeterministic computation

Max height (longest path) $t(n)$

Max # of nodes

$b = \text{branching per level}$

$b^{t(n)} = 2^{O(t(n))}$
Flashback: Nondet $\Rightarrow$ Deterministic TM: Space

$D$

Always has input, never changes: $n$ space

Used to run each path (re-copy input here for each path): $t(n)$ space

Tracks which node we are on, $2^{O(t(n))}$ (exponential) space??

3 tapes
Nondet $\Rightarrow$ Deterministic TM: Space

Let $N$ be an NTM deciding language $A$ in space $f(n)$

- This means a single path could use $f(n)$ space
- That path could take $2^{df(n)}$ steps
  - (That’s the possible ways to fill the space)
  - Each step could be a non-deterministic branch that must be saved
- So naively tracking these branches requires $2^{df(n)}$ space!

- Instead, let’s “divide and conquer” to reduce space!
“Divide and Conquer” TM Config Sequences

• Want to check whether:

\[ 2^{O(f(n))} \text{ steps (possibly branching)} \]

\[ C_{\text{start}} \rightarrow C_{\text{accept}} \]

Remembering the branch at every step costs exponential space.

\[ \log(2^{O(f(n))}) = O(f(n)) \]

Number of splits:

Each split must remember a “\(c_m\)” config = \(O(f(n))\) space

• Instead, we check whether:

\[ 2^{O(f(n))/2} \text{ steps} \]

\[ C_{\text{start}} \rightarrow C_{m} \rightarrow C_{\text{accept}} \]

Remembering these steps costs half the space …

So long as we save the intermediate config … and we can reuse that space to check the second half

Total:

\[ O(f(n)) \times O(f(n)) = O(f^2(n)) \text{ space} \]

(Savitch’s Thm)

• Keep dividing …

\[ C_{\text{start}} \rightarrow \rightarrow \rightarrow \rightarrow C_{\text{accept}} \]
Formally: A “Yielding” Algorithm

\[ \text{CANYIELD = “On input } c_1, c_2, \text{ and } t:\]

1. If \( t = 1 \), then test directly whether \( c_1 = c_2 \) or whether \( c_1 \) yields \( c_2 \) in one step according to the rules of \( N \). \text{Accept} if either test succeeds; \text{reject} if both fail.
2. If \( t > 1 \), then for each configuration \( c_m \) of \( N \) using space \( f(n) \):
   3. Run CANYIELD\( (c_1, c_m, \frac{t}{2}) \).
   4. Run CANYIELD\( (c_m, c_2, \frac{t}{2}) \).
5. If steps 3 and 4 both accept, then \text{accept}.
6. If haven’t yet accepted, \text{reject}.”

What’s the middle config? Try them all (it doesn’t use any more space, per loop)

“divide and conquer”
Savitch’s Theorem: Proof

• Let $N$ be an NTM deciding language $A$ in space $f(n)$
• Construct equivalent deterministic TM $M$ using $O(f^2(n))$ space:

$$M = \text{“On input } w:\n1. \text{ Output the result of CANYIELD}(c_{\text{start}}, c_{\text{accept}}, 2^{df(n)})\text{.”}$$

• $c_{\text{start}} = \text{start configuration of } N$
• $c_{\text{accept}} = \text{new accepting config where all } N's \text{ accepting configs go}$
PSPACE

DEFINITION

PSPACE is the class of languages that are decidable in polynomial space on a deterministic Turing machine. In other words,

$$PSPACE = \bigcup_{k} SPACE(n^k).$$
**NPSPACE**

Analogous to $P$ and $NP$ for time complexity

**DEFINITION**

$NPSPACE$ is the class of languages that are decidable in polynomial space on a deterministic Turing machine. In other words,

$$NPSPACE = \bigcup_{k} NSPACE(n^k).$$

But $P \subseteq PSPACE$ and $NP \subseteq NPSPACE$
- Because each step can use at most one extra tape cell
- But space can be re-used
Flashback: Does P = NP?

Proving P ≠ NP is hard because how do you prove an algorithm doesn’t have a poly time algorithm? (in general it’s hard to prove that something doesn’t exist)
PSPACE = NPSPACE?

- **PSPACE:** langs decidable in poly space on deterministic TM

- **NPSPACE:** langs decidable in poly space on nondeterministic TM

**Theorem:** PSPACE = NPSPACE !!!

**Proof:** By Savitch’s Theorem!

---

**THEOREM**

Savitch’s theorem For any function $f: N \to R^+$, where $f(n) \geq n$, $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$. 
Space vs Time

• \( P \subseteq \text{PSPACE} \) and \( \text{NP} \subseteq \text{NPSPACE} \)
  • Because each step can use at most one extra tape cell
  • And space can be re-used

• \( \text{PSPACE} \subseteq \text{EXPTIME} \)
  • Because an \( f(n) \) space TM has \( 2^{O(f(n))} \) possible configurations
  • And a halting TM cannot repeat a configuration

• We already know \( P \subseteq \text{NP} \) and \( \text{PSPACE} = \text{NPSPACE} \) ... so:

\[ P \subseteq \text{NP} \subseteq \text{PSPACE} = \text{NPSPACE} \subseteq \text{EXPTIME} \]
Space vs Time: Conjecture

Researchers believe these are all completely contained within each other.

But this is an open conjecture!

The only progress so far is: \( P \subseteq \text{EXPTIME} \) (we will prove next week).

\[ P \subseteq NP \subseteq \text{PSPACE} = \text{NPSPACE} \subseteq \text{EXPTIME} \]
**Review: NP-Completeness**

**Definition**

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

1. $B$ is in NP, and
2. every $A$ in NP is polynomial time reducible to $B$.

The reduction must be “easy”

These are the “hardest” problems (in NP) to solve

Potentially helps answer $P=NP$? question

**Theorem**

If $B$ is NP-complete and $B \in P$, then $P = NP$.
**NP-Completeness vs NP-Hardness**

**DEFINITION**

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

1. $B$ is in NP, and
2. every $A$ in NP is polynomial time reducible to $B$.

"NP-Hard"

"NP-Complete" = in NP + "NP-Hard"

So a language can be NP-hard but not NP-complete!
The Halting Problem is **NP-Hard**

\[ \text{HALT}_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w \} \]

**Proof:** Reduce **3SAT** to the Halting Problem

(Why does this prove that the Halting Problem is **NP-hard**?)

Because **3SAT** is **NP-complete**! (so every **NP** problem is poly time reducible to **3SAT**)
The Halting Problem is **NP-Hard**

\[ HALT^\text{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w \} \]

**Computable function, from 3SAT → HALT^\text{TM}:**

On input \( \phi \), a formula in 3cnf:

- **Construct TM** \( M \)
  
  \[ M = \text{on input } \phi \]
  
  - Try all assignments
    - If any satisfy \( \phi \), then accept
  - When all assignments have been tried, **start over**

- **Output** \( \langle M, \phi \rangle \)
  
  \[ \Rightarrow \text{If } \phi \text{ has a satisfying assignment, then } M \text{ halts on } \phi \]
  
  \[ \Leftarrow \text{If } \phi \text{ has no satisfying assignment, then } M \text{ loops on } \phi \]
A language $B$ is **NP-complete** if it satisfies two conditions:

1. $B$ is in NP, and
2. every $A$ in NP is polynomial time reducible to $B$.

So a language can satisfy only condition #2.
**Definition**

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

1. $B$ is in NP, and
2. every $A$ in NP is polynomial time reducible to $B$.

So a language can satisfy only condition #2

Can a language satisfy only condition #1?

Yes, every language in $P$ ...

(... unless $P = NP$)

Can a non-$P$ language satisfy only condition #1?

Yes ...

... but that implies $P \neq NP$, so it’s not known for sure
PSPACE-Completeness

DEFINITION

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

1. $B$ is in PSPACE, and
2. every $A$ in PSPACE is polynomial time reducible to $B$.

If $B$ merely satisfies condition 2, we say that it is **PSPACE-hard**.

Condition #2 hard to prove the first time

The reduction must still be “easy”
Flashback: NP-Completeness

**Definition**

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

1. $B$ is in NP, and
2. every $A$ in NP is polynomial time reducible to $B$.

**Theorem**

$SAT$ is NP-complete.

$SAT = \{ \langle \phi \rangle | \phi \text{ is a satisfiable Boolean formula} \}$
**PSPACE-Completeness**

**Definition**

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

1. $B$ is in PSPACE, and
2. every $A$ in PSPACE is polynomial time reducible to $B$.

If $B$ merely satisfies condition 2, we say that it is **PSPACE-hard**.

**Theorem**

The first PSPACE-complete problem:

$TQBF$ is PSPACE-complete.

$TQBF = \{ \langle \phi \rangle \mid \phi \text{ is a true fully quantified Boolean formula} \}$
$TQBF$ = \{ \langle \phi \rangle | \phi \text{ is a true fully quantified Boolean formula} \}$
# Flashback: Boolean Formulas

<table>
<thead>
<tr>
<th>A Boolean ______</th>
<th>Is ...</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>TRUE or FALSE (or 1 or 0)</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>Variable</td>
<td>Represents a Boolean value</td>
<td>x, y, z</td>
</tr>
<tr>
<td>Operation</td>
<td>Combines Boolean variables</td>
<td>AND, OR, NOT ((\land, \lor, \text{ and } \neg))</td>
</tr>
<tr>
<td>Formula (\phi)</td>
<td>Combines vars and operations</td>
<td>((\overline{x} \land y) \lor (x \land \overline{z}))</td>
</tr>
<tr>
<td>Literal</td>
<td>A var or a negated var</td>
<td>(x, \text{ or } \overline{x})</td>
</tr>
<tr>
<td>Clause</td>
<td>Literals ORed together</td>
<td>((x_1 \lor \overline{x}_2 \lor \overline{x}_3 \lor x_4))</td>
</tr>
</tbody>
</table>
Flashback: The Language of Math Statements

1. $\forall q \exists p \forall x,y \left[ p > q \land (x,y > 1 \rightarrow xy \neq p) \right]$, 
2. $\forall a,b,c,n \left[ (a,b,c > 0 \land n > 2) \rightarrow a^n + b^n \neq c^n \right]$, and 
3. $\forall q \exists p \forall x,y \left[ p > q \land (x,y > 1 \rightarrow (xy \neq p \land xy \neq p+2)) \right]$
Flashback: Mathematical Statements Alphabet

• Strings in the language are drawn from the following chars:

  • $\land$, $\lor$, $\neg$  \hspace{1cm} Boolean operations
  • ($,$), $[$, $]$  \hspace{1cm} parentheses
  • $\forall$, $\exists$  \hspace{1cm} quantifiers
  • $x$  \hspace{1cm} variables
  • $R_1, \ldots, R_k$  \hspace{1cm} Relation symbols
Flashback: Formulas and Sentences

• A mathematical statement is well-formed, i.e., a formula, if it’s:
  • an atomic formula: $R_i(x_1, ..., x_k)$
  • $\phi_1 \land \phi_2$
  • $\phi_1 \lor \phi_2$
  • $\neg \phi$
    • where $\phi, \phi_1,$ and $\phi_2$ are formulas
  • $\forall x \,[ \phi ]$
  • $\exists x \,[ \phi ]$
    • where $\phi$ is a formula
    • $x$’s “scope” is in the following brackets
    • A free variable is a variable that is outside the scope of a quantifier

• A sentence is a formula with no free variables
Flashback: Universes, Models, and Theories

• A **universe** is the set of values that variables can represent
  • E.g., the universe of the natural numbers
  • **Boolean Formulas** use values from the universe of \{True, False\}

• A **model** is:
  1. a universe, and
  2. an assignment of relations to relation symbols, e.g., AND, OR, NOT

• A **theory** is the set of all **true sentences** in a model’s language
# Quantified Boolean Formulas

<table>
<thead>
<tr>
<th>A Boolean ______</th>
<th>Is ...</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>TRUE or FALSE (or 1 or 0)</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>Variable</td>
<td>Represents a Boolean value</td>
<td>$x, y, z$</td>
</tr>
<tr>
<td>Operation</td>
<td>Combines Boolean variables</td>
<td>AND, OR, NOT ($\land, \lor$, and $\neg$)</td>
</tr>
<tr>
<td>Formula $\phi$</td>
<td>Combines vars and operations</td>
<td>$(\overline{x} \land y) \lor (x \land \overline{z})$</td>
</tr>
<tr>
<td>Literal</td>
<td>A var or a negated var</td>
<td>$x$ or $\overline{x}$, $x_1 \lor \overline{x}_2 \lor \overline{x}_3 \lor x_4$</td>
</tr>
<tr>
<td>Clause</td>
<td>Literals ORed together</td>
<td>$\phi = \forall x \exists y \left[ (x \lor y) \land (\overline{x} \lor \overline{y}) \right]$</td>
</tr>
</tbody>
</table>

Quantifiers: $\exists$ or $\forall$

Quantified Formula: Formula with quantifiers

Fully Quantified Formula: Sentence, no free vars
THEOREM

$TQBF$ is PSPACE-complete.

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

DEFINITION

A language $B$ is \textbf{PSPACE-complete} if it satisfies two conditions:

1. $B$ is in PSPACE, and
2. every $A$ in PSPACE is polynomial time reducible to $B$.

If $B$ merely satisfies condition 2, we say that it is \textbf{PSPACE-hard}. 
**TQBF is in PSPACE**

Let $m = \# \text{ variables in formula}$

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

**Proof**

First, we give a polynomial space algorithm deciding $TQBF$.

$T = \text{“On input } \langle \phi \rangle, \text{ a fully quantified Boolean formula:}$$\begin{align*}
1. & \text{ If } \phi \text{ contains no quantifiers, then it is an expression with only } \\
& \text{ constants, so evaluate } \phi \text{ and } \text{accept if it is true; otherwise, } \text{reject.} \\
2. & \text{ If } \phi \text{ equals } \exists x \, \psi, \text{ recursively call } T \text{ on } \psi, \text{ first with 0 substituted } \\
& \text{ for } x \text{ and then with 1 substituted for } x. \text{ If either result is accept, } \\
& \text{ then } \text{accept; otherwise, } \text{reject.} \\
3. & \text{ If } \phi \text{ equals } \forall x \, \psi, \text{ recursively call } T \text{ on } \psi, \text{ first with 0 substituted } \\
& \text{ for } x \text{ and then with 1 substituted for } x. \text{ If both results are accept, } \\
& \text{ then } \text{accept; otherwise, } \text{reject.”}
\end{align*}$

At most $m$ recursive calls, so $O(m)$ space
THEOREM

$TQBF$ is PSPACE-complete.

\[
TQBF = \{ \langle \phi \rangle \mid \phi \text{ is a true fully quantified Boolean formula} \}
\]

DEFINITION

A language $B$ is $\text{PSPACE-complete}$ if it satisfies two conditions:

1. $B$ is in PSPACE, and
2. every $A$ in PSPACE is polynomial time reducible to $B$.

If $B$ merely satisfies condition 2, we say that it is $\text{PSPACE-hard}$.
**TQBF** is **PSPACE**-Hard

\[ TQBF = \{ \langle \phi \rangle \mid \phi \text{ is a true fully quantified Boolean formula} \} \]

**Idea:** Imitate Cook-Levin Theorem
Flashback: \( SAT \) is NP-complete

- **Proof idea:**
  - Give an algorithm that reduces accepting tableaus to satisfiable formulas

- Thus every string in the NP lang will be mapped to a sat. formula
  - and vice versa

\[
SAT = \{ \langle \phi \rangle | \phi \text{ is a satisfiable Boolean formula} \}
\]
An \textbf{NP} “Tableau”

- input $w = w_1 \ldots w_n$

- At most $n^k$ configs
  - (why?)

- Each config has length $n^k$
  - (why?)
A PSPACE "Tableau"

- input $w = w_1 \ldots w_n$

- At most $2^{O(n^k)}$ configs
  - (why?)

- Each config has length $n^k$
  - (why?)

$\phi_{\text{cell}} \land \phi_{\text{start}} \land \phi_{\text{move}} \land \phi_{\text{accept}}$

Converting this to a formula would take exponential space and time!
$TQBF$ is **PSPACE**-Hard

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

**Another Idea:** use quantifiers to “divide and conquer”

(like we did for Savitch’s Theorem)

Let $f(n) = n^k$ be the space usage of the TM
Recursively Defined Formulas: Try # 1

\[ \phi_{c_1,c_2,t} = \exists m_1 \left[ \phi_{c_1,m_1,\frac{t}{2}} \land \phi_{m_1,c_2,\frac{t}{2}} \right] \]

*t halved, but formula doubles in size (two subformulas)*

Doesn’t work! Still exponential!
Recursively Defined Formulas: Try # 2

\[
\phi_{c_1, c_2, t} = \exists m_1 \left[ \phi_{c_1, m_1, \frac{t}{2}} \land \phi_{m_1, c_2, \frac{t}{2}} \right]
\]

\[
\phi_{c_1, c_2, t} = \exists m_1 \forall (c_3, c_4) \in \{(c_1, m_1), (m_1, c_2)\} \left[ \phi_{c_3, c_4, \frac{t}{2}} \right]
\]

What's this?

Use \(\forall\) quantifier to consolidate formula size

\(\forall x \in \{y, z\} \ldots\) Shorthand for \(\forall x \left[ (x = y \lor x = z) \implies \ldots \right]\)

(And \(=, \implies\) can be converted to \(\land\) AND and \(\lor\) OR)

\(t = 2^{O(f(n))}\),

so # halvings (subformulas) = \(\log(2^{O(f(n))}) = O(f(n))\)

What do the base case subformulas look like?

t halved, only 1 subformula
Recursively Defined Formulas: Base Case

\[ \phi_{c_1, c_2, t} \]

This formula must encode that \( c_1 \rightarrow c_2 \) is a valid TM step ...

... using the same encoding as Cook-Levin!

Size of this subformula = \( O(f(n)) \)

Total size of all subformulas = \( O(f(n)) \times O(f(n)) = O(f^2(n)) \)
**TQBF is PSPACE-Hard**

\[ TQBF = \{ \langle \phi \rangle \mid \phi \text{ is a true fully quantified Boolean formula} \} \]

**Another Idea: use quantifiers to “divide and conquer”**

(like we did for Savitch’s Theorem)

- If \( M \) accepts \( w \), then the formula is TRUE
  - Because formula encodes accepting config seqs
- If \( M \) rejects \( w \), then the formula is FALSE
  - Because there’s no config seq reaching accept state

Let \( f(n) = n^k \) be the space usage of the TM \( M \) deciding some language \( A \)

\[ 2^{O(n^k)} \]
**THEOREM**

$TQBF$ is PSPACE-complete.

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

---

**DEFINITION**

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

1. $B$ is in PSPACE, and
2. every $A$ in PSPACE is polynomial time reducible to $B$.

If $B$ merely satisfies condition 2, we say that it is **PSPACE-hard**.

i.e., the “hardest” problem in PSPACE
Check-in Quiz 11/29

On gradescope