COMP2210: Theory of Computing

Lecture 24: Space Complexity

Corina Cîrstea
Outline

1. Space Complexity
2. PSPACE Completeness
Recall: The time complexity of a decider is the function \( f : \mathbb{N} \rightarrow \mathbb{N} \) where \( f(n) \) is the maximum number of steps the machine uses on any branch of its computation, on any input of length \( n \).

Definition

The space complexity of a decider is the function \( f : \mathbb{N} \rightarrow \mathbb{N} \) where \( f(n) \) is the maximum number of tape cells the machine scans on any branch of its computation, on any input of length \( n \).
A Useful Result

**Theorem**

Assume $f(n) \geq n$. If a decider runs in $f(n)$ space, then it runs in $2^{O(f(n))}$ time.

**Proof sketch.**

- there are $|Q| \cdot f(n) \cdot 2^{O(f(n))}$ possible configurations;
  - $|Q|$ states
  - $f(n)$ possible positions for the tape head
  - $2^{O(f(n))}$ possibilities for the tape contents
- a decider will not repeat configurations;
- for $f(n) \geq n$, $f(n) \cdot 2^{O(f(n))}$ is $2^{O(f(n))}$. 

□
Example: SAT

**Recall:** We believe we cannot solve SAT with a polynomial **time** algorithm. How about polynomial space?

Here is a (deterministic!) polynomial space algorithm for SAT:

| On input \( \phi \), with \( \phi \) a boolean formula in variables \( x_1, \ldots, x_n \):
| --- |
| 1. For each assignment to the variables \( x_1, \ldots, x_n \):
  | • evaluate \( \phi \) on that assignment |
| 2. If \( \phi \) ever evaluates to 1 then accept, otherwise reject once all assignments were explored. |

Above runs in polynomial space as it can **reuse space:**

- only the current truth assignment needs to be stored, and this can be done with \( O(n) \) space!
- evaluating the expression can be done with \( O(m) \) space, where \( m \) is the size of the expression.
### Definition

The class **PSPACE** is the set of all decision problems that can be solved by a *deterministic* Turing machine using polynomial space.

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>The class <strong>PSPACE</strong> is the set of all decision problems that can be solved by a <em>deterministic</em> Turing machine using polynomial space.</td>
</tr>
</tbody>
</table>

### Definition

The class **NPSPACE** is the set of all decision problems that can be solved by a *non-deterministic* Turing machine using polynomial space.

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>The class <strong>NPSPACE</strong> is the set of all decision problems that can be solved by a <em>non-deterministic</em> Turing machine using polynomial space.</td>
</tr>
</tbody>
</table>
Anything in PSPACE is in NPSPACE. If a problem can be solved in polynomial space by a deterministic machine, then also by a non-deterministic one:
**Recall:** Every $t(n)$ time non-deterministic Turing machine has an equivalent $2^{O(t(n))}$ time deterministic Turing machine.

Things are different when it comes to space:

**Theorem (Savitch’s Theorem)**

For $f(n) \geq n$, every $f(n)$ space non-deterministic Turing machine has an equivalent $f(n)^2$ space deterministic Turing machine.

**Corollary**

$PSPACE = NPSPACE$. 
Proof Idea (not examinable!)

- naive approach (straightforward simulation of the non-deterministic TM with a deterministic one) doesn’t work
  - the simulation needs to keep track of the branch it is currently exploring (the choices made)
  - an $f(n)$-space branch can have up to $2^{O(f(n))}$ non-deterministic steps, which need to be recorded
- want to know if the non-deterministic TM can go from the initial configuration to an accepting configuration
- use divide and conquer approach by reducing to the yieldability problem:
  - yieldability problem: given NTM configurations $c_1$ and $c_2$ and $t > 0$, can the NTM get from $c_1$ to $c_2$ within $t$ steps?
  - solve the yieldability problem with $c_1$ the starting configuration, $c_2$ the only accepting configuration (need to modify the machine for this!), and $t$ the maximum number of steps the machine can use;
  - recursive algorithm for the yieldability problem takes $O(f(n)^2)$ space (as we can reuse space!).

(More details in Sipser pp. 310-311.)
The complexity class EXPTIME

So far we defined the classes P, NP, PSPACE and NPSPACE.

Definition

The class EXPTIME is the set of all decision problems that can be solved by a deterministic Turing machine in time $O(2^{n^k})$ for some $k \in \mathbb{N}$. 
Relationships between complexity classes

- **P \subseteq PSPACE**: a machine that runs in polynomial time can only use polynomial space.
- **NP \subseteq NPSPACE**: same reason.
- Hence, using Savitch’s theorem, **NP \subseteq PSPACE**.
- **PSPACE \subseteq EXPTIME**:
  - If a decider runs in \( f(n) \) space, then it runs in \( 2^{O(f(n))} \) time.
We don’t know which of these inclusions are strict!

All we know is \( P \subsetneq \text{EXPTIME} \) \((\subsetneq \) denotes strict subset).

**Note:** One can also define EXPSPACE. We know that

\[
\text{EXPTIME} \subseteq \text{EXPSPACE} \\
\text{PSPACE} \subsetneq \text{EXPSPACE}
\]
PSPACE-Completeness

Definition

A decision problem $X$ is PSPACE-complete if

- $X$ is in PSPACE
- every problem $A$ in PSPACE is polynomial-time reducible to $X$

If $X$ only satisfies the second condition, it is called PSPACE-hard.

Why not polynomial-space reducible?

- we defined NP-complete problems by using polynomial-time reductions . . .
- . . . because we wanted the reduction to be easy compared to typical problems in NP!
- if we used polynomial space reduction here, then an easy solution for the problem we are reducing to would not necessarily give an easy solution to the problem we are reducing from!
A PSPACE-Complete Problem: TQBF

Let **TQBF** be the problem of deciding whether a fully quantified boolean formula is true.

Examples of fully quantified boolean formulas:

- $\forall x \forall y (x \lor y)$
- $\forall x \exists y ((x \lor y) \land (\neg x \lor \neg y))$

**Exercise:** Show that TQBF is in PSPACE.

**Theorem**

*TQBF is PSPACE-complete.*
Geography Game

- players take turn naming cities
- chosen city must begin with the same letter that ended the previous city
- start with a designated city, no repetitions allowed
- can model this using a graph $G$: nodes are cities, edges are valid moves
- the problem: does the first player have a winning strategy for the game $G$ starting at node $b$?
- this problem is in PSPACE
PSPACE-Hard Problems

- **generalised geography game**: played on an arbitrary directed graph with designated start node
  - similar requirement: the chosen path must be *simple* (does not repeat nodes)
  - also in PSPACE, and PSPACE-complete!
  - the number of moves is *polynomial* in the size of the board

- **generalised chess**: PSPACE-hard, believed not to be in PSPACE

- **GO**: PSPACE-hard, believed not to be in PSPACE

- lots more . . .
Summary

- (N)PSPACE is the class of problems that can be solved with polynomial space on a (non-)deterministic Turing machine.
- PSPACE = NPSPACE (Savitch’s theorem).
- PSPACE-complete = PSPACE-hard \cap PSPACE
- TQBF is an example of a PSPACE-complete problem.