

COMPLEXITY THEORY

Lecture 9: Space Complexity and PSPACE

Stephan Mennicke

Knowledge-Based Systems

TU Dresden, 10 Nov 2025

More recent versions of this slide deck might be available.
For the most current version of this course, see
https://iccl.inf.tu-dresden.de/web/Complexity_Theory/en

Review

Review: Space Complexity Classes

Recall our earlier definitions of space complexities:

Definition 9.1: Let $f : \mathbb{N} \rightarrow \mathbb{R}^+$ be a function.

- (1) **DSpace**($f(n)$) is the class of all languages L for which there is an $O(f(n))$ -space bounded Turing machine deciding L .
- (2) **NSpace**($f(n)$) is the class of all languages L for which there is an $O(f(n))$ -space bounded nondeterministic Turing machine deciding L .

Being $O(f(n))$ -space bounded requires a (nondeterministic) TM

- to halt on every input and
- to use $\leq f(|w|)$ tape cells on every computation path.

Space Complexity Classes

Some important space complexity classes:

$$L = \text{LogSpace} = \text{DSpace}(\log n)$$

logarithmic space

$$\text{PSpace} = \bigcup_{d \geq 1} \text{DSpace}(n^d)$$

polynomial space

$$\text{ExpSpace} = \bigcup_{d \geq 1} \text{DSpace}(2^{n^d})$$

exponential space

$$\text{NL} = \text{NLogSpace} = \text{NSpace}(\log n)$$

nondet. logarithmic space

$$\text{NPSpace} = \bigcup_{d \geq 1} \text{NSpace}(n^d)$$

nondet. polynomial space

$$\text{NExpSpace} = \bigcup_{d \geq 1} \text{NSpace}(2^{n^d})$$

nondet. exponential space

The Power of Space

Space seems to be more powerful than time
because space can be reused.

The Power of Space

Space seems to be more powerful than time
because space can be reused.

Example 9.2: SAT can be solved in linear space:

Just iterate over all possible truth assignments (each linear in size) and check if one satisfies the formula.

The Power of Space

Space seems to be more powerful than time because **space can be reused**.

Example 9.2: SAT can be solved in linear space:

Just iterate over all possible truth assignments (each linear in size) and check if one satisfies the formula.

Example 9.3: TAUTOLOGY can be solved in linear space:

Just iterate over all possible truth assignments (each linear in size) and check if all satisfy the formula.

The Power of Space

Space seems to be more powerful than time
because **space can be reused**.

Example 9.2: SAT can be solved in linear space:

Just iterate over all possible truth assignments (each linear in size) and check if one satisfies the formula.

Example 9.3: TAUTOLOGY can be solved in linear space:

Just iterate over all possible truth assignments (each linear in size) and check if all satisfy the formula.

More generally: $NP \subseteq PSpace$ and $coNP \subseteq PSpace$

Linear Compression

Theorem 9.4: For every function $f : \mathbb{N} \rightarrow \mathbb{R}^+$, for all $c \in \mathbb{N}$, and for every f -space bounded (deterministic/nondeterministic) Turing machine $\mathcal{M}$:
there is a $\max\{1, \frac{1}{c}f(n)\}$ -space bounded (deterministic/nondeterministic) Turing machine $\mathcal{M}'$ that accepts the same language as $\mathcal{M}$.

Linear Compression

Theorem 9.4: For every function $f : \mathbb{N} \rightarrow \mathbb{R}^+$, for all $c \in \mathbb{N}$, and for every f -space bounded (deterministic/nondeterministic) Turing machine $\mathcal{M}$:
there is a $\max\{1, \frac{1}{c}f(n)\}$ -space bounded (deterministic/nondeterministic) Turing machine $\mathcal{M}'$ that accepts the same language as $\mathcal{M}$.

Proof idea: Similar to (but much simpler than) linear speed-up. □

Linear Compression

Theorem 9.4: For every function $f : \mathbb{N} \rightarrow \mathbb{R}^+$, for all $c \in \mathbb{N}$, and for every f -space bounded (deterministic/nondeterministic) Turing machine $\mathcal{M}$:
there is a $\max\{1, \frac{1}{c}f(n)\}$ -space bounded (deterministic/nondeterministic) Turing machine $\mathcal{M}'$ that accepts the same language as $\mathcal{M}$.

Proof idea: Similar to (but much simpler than) linear speed-up. □

This justifies using O -notation for defining space classes.

Tape Reduction

Theorem 9.5: For every function $f : \mathbb{N} \rightarrow \mathbb{R}^+$ all $k \geq 1$ and $\mathbf{L} \subseteq \Sigma^*$:

If $\mathbf{L}$ can be decided by an f -space bounded k -tape Turing-machine,
then it can also be decided by an f -space bounded 1-tape Turing-machine.

Tape Reduction

Theorem 9.5: For every function $f : \mathbb{N} \rightarrow \mathbb{R}^+$ all $k \geq 1$ and $\mathbf{L} \subseteq \Sigma^*$:

If $\mathbf{L}$ can be decided by an f -space bounded k -tape Turing-machine, then it can also be decided by an f -space bounded 1-tape Turing-machine.

Proof idea: Combine tapes with a similar reduction as for time. Compress space to avoid linear increase. □

Note: We still use a separate read-only input tape to define some space complexities, such as LogSpace.

Time vs. Space

Theorem 9.6: For all functions $f : \mathbb{N} \rightarrow \mathbb{R}^+$:

$$\text{DTime}(f) \subseteq \text{DSpace}(f) \quad \text{and} \quad \text{NTime}(f) \subseteq \text{NSpace}(f)$$

Proof: Visiting a cell takes at least one time step. □

Time vs. Space

Theorem 9.6: For all functions $f : \mathbb{N} \rightarrow \mathbb{R}^+$:

$$\text{DTime}(f) \subseteq \text{DSpace}(f) \quad \text{and} \quad \text{NTime}(f) \subseteq \text{NSpace}(f)$$

Proof: Visiting a cell takes at least one time step. □

Theorem 9.7: For all functions $f : \mathbb{N} \rightarrow \mathbb{R}^+$ with $f(n) \geq \log n$:

$$\text{DSpace}(f) \subseteq \text{DTime}(2^{O(f)}) \quad \text{and} \quad \text{NSpace}(f) \subseteq \text{DTime}(2^{O(f)})$$

Time vs. Space

Theorem 9.6: For all functions $f : \mathbb{N} \rightarrow \mathbb{R}^+$:

$$\text{DTime}(f) \subseteq \text{DSpace}(f) \quad \text{and} \quad \text{NTime}(f) \subseteq \text{NSpace}(f)$$

Proof: Visiting a cell takes at least one time step. □

Theorem 9.7: For all functions $f : \mathbb{N} \rightarrow \mathbb{R}^+$ with $f(n) \geq \log n$:

$$\text{DSpace}(f) \subseteq \text{DTime}(2^{O(f)}) \quad \text{and} \quad \text{NSpace}(f) \subseteq \text{DTime}(2^{O(f)})$$

Proof: Based on configuration graphs and a bound on the number of possible configurations.

Number of Possible Configurations

Let $\mathcal{M} := (Q, \Sigma, \Gamma, q_0, \delta, q_{\text{start}})$ be a 2-tape Turing machine
(1 read-only input tape + 1 work tape)

Recall: A configuration of $\mathcal{M}$ is a quadruple (q, p_1, p_2, x) where

- $q \in Q$ is the current state,
- $p_i \in \mathbb{N}$ is the head position on tape i , and
- $x \in \Gamma^*$ is the tape content.

Let $w \in \Sigma^*$ be an input to $\mathcal{M}$ and $n := |w|$.

- Then also $p_1 \leq n$.
- If $\mathcal{M}$ is $f(n)$ -space bounded we can assume $p_2 \leq f(n)$ and $|x| \leq f(n)$

Number of Possible Configurations

Let $\mathcal{M} := (Q, \Sigma, \Gamma, q_0, \delta, q_{\text{start}})$ be a 2-tape Turing machine
(1 read-only input tape + 1 work tape)

Recall: A configuration of $\mathcal{M}$ is a quadruple (q, p_1, p_2, x) where

- $q \in Q$ is the current state,
- $p_i \in \mathbb{N}$ is the head position on tape i , and
- $x \in \Gamma^*$ is the tape content.

Let $w \in \Sigma^*$ be an input to $\mathcal{M}$ and $n := |w|$.

- Then also $p_1 \leq n$.
- If $\mathcal{M}$ is $f(n)$ -space bounded we can assume $p_2 \leq f(n)$ and $|x| \leq f(n)$

Hence, there are at most

$$|Q| \cdot n \cdot f(n) \cdot |\Gamma|^{f(n)} = n \cdot 2^{O(f(n))} = 2^{O(f(n))}$$

different configurations on inputs of length n (the last equality requires $f(n) \geq \log n$).

Configuration Graphs

The possible computations of a TM $\mathcal{M}$ (on input w) form a directed graph:

- Vertices: configurations that $\mathcal{M}$ can reach (on input w)
- Edges: there is an edge from C_1 to C_2 if $C_1 \vdash_{\mathcal{M}} C_2$
(C_2 reachable from C_1 in a single step)

This yields the **configuration graph**:

- Could be infinite in general.
- For $f(n)$ -space bounded 2-tape TMs,
there can be at most $2^{O(f(n))}$ vertices and $(2^{O(f(n))})^2 = 2^{O(f(n))}$ edges

Configuration Graphs

The possible computations of a TM $\mathcal{M}$ (on input w) form a directed graph:

- Vertices: configurations that $\mathcal{M}$ can reach (on input w)
- Edges: there is an edge from C_1 to C_2 if $C_1 \vdash_{\mathcal{M}} C_2$
(C_2 reachable from C_1 in a single step)

This yields the **configuration graph**:

- Could be infinite in general.
- For $f(n)$ -space bounded 2-tape TMs,
there can be at most $2^{O(f(n))}$ vertices and $(2^{O(f(n))})^2 = 2^{O(f(n))}$ edges

A **computation** of $\mathcal{M}$ on input w corresponds to a **path** in the configuration graph from the **start** configuration to a **stop** configuration.

Hence, to test if $\mathcal{M}$ accepts input w ,

- construct the configuration graph and
- find a path from the start to an accepting stop configuration.

Time vs. Space

Theorem 9.6: For all functions $f : \mathbb{N} \rightarrow \mathbb{R}^+$:

$$\text{DTime}(f) \subseteq \text{DSpace}(f) \quad \text{and} \quad \text{NTime}(f) \subseteq \text{NSpace}(f)$$

Proof: Visiting a cell takes at least one time step. □

Theorem 9.7: For all functions $f : \mathbb{N} \rightarrow \mathbb{R}^+$ with $f(n) \geq \log n$:

$$\text{DSpace}(f) \subseteq \text{DTime}(2^{O(f)}) \quad \text{and} \quad \text{NSpace}(f) \subseteq \text{DTime}(2^{O(f)})$$

Proof: Build the configuration graph (time $2^{O(f(n))}$) and find a path from the start to an accepting stop configuration (time $2^{O(f(n))}$). □

Basic Space/Time Relationships

Applying the results of the previous slides, we get the following relations:

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSpace \subseteq NPSpace \subseteq ExpTime \subseteq NExpTime$$

We also noted $P \subseteq coNP \subseteq PSpace$.

Open questions:

- What is the relationship between space classes and their co-classes?
- What is the relationship between deterministic and non-deterministic space classes?

Nondeterminism in Space

Most experts think that nondeterministic TMs can solve strictly more problems when given the same amount of time than a deterministic TM:

Most believe that $P \subsetneq NP$

How about nondeterminism in space-bounded TMs?

Nondeterminism in Space

Most experts think that nondeterministic TMs can solve strictly more problems when given the same amount of time than a deterministic TM:

Most believe that $P \subsetneq NP$

How about nondeterminism in space-bounded TMs?

Theorem 9.8 (Savitch's Theorem, 1970): For any function $f : \mathbb{N} \rightarrow \mathbb{R}^+$ with $f(n) \geq \log n$:

$$\text{NSpace}(f(n)) \subseteq \text{DSpace}(f^2(n)).$$



That is: nondeterminism adds **almost** no power to space-bounded TMs!

Consequences of Savitch's Theorem

Theorem 9.8 (Savitch's Theorem, 1970): For any function $f : \mathbb{N} \rightarrow \mathbb{R}^+$ with $f(n) \geq \log n$:

$$\text{NSpace}(f(n)) \subseteq \text{DSpace}(f^2(n)).$$

Consequences of Savitch's Theorem

Theorem 9.8 (Savitch's Theorem, 1970): For any function $f : \mathbb{N} \rightarrow \mathbb{R}^+$ with $f(n) \geq \log n$:

$$\text{NSpace}(f(n)) \subseteq \text{DSpace}(f^2(n)).$$

Corollary 9.9: $\text{PSpace} = \text{NPSpace}$.

Proof: $\text{PSpace} \subseteq \text{NPSpace}$ is clear. The converse follows since the square of a polynomial is still a polynomial. □

Similarly for “bigger” classes, e.g., $\text{ExpSpace} = \text{NExpSpace}$.

Consequences of Savitch's Theorem

Theorem 9.8 (Savitch's Theorem, 1970): For any function $f : \mathbb{N} \rightarrow \mathbb{R}^+$ with $f(n) \geq \log n$:

$$\text{NSpace}(f(n)) \subseteq \text{DSpace}(f^2(n)).$$

Corollary 9.9: $\text{PSpace} = \text{NPSpace}$.

Proof: $\text{PSpace} \subseteq \text{NPSpace}$ is clear. The converse follows since the square of a polynomial is still a polynomial. □

Similarly for “bigger” classes, e.g., $\text{ExpSpace} = \text{NExpSpace}$.

Corollary 9.10: $\text{NL} \subseteq \text{DSpace}(O(\log^2 n))$.

Note that $\log^2(n) \notin O(\log n)$, so we do not obtain $\text{NL} = \text{L}$ from this.

Proving Savitch's Theorem

Simulating nondeterminism with more space:

- Use configuration graph of nondeterministic space-bounded TM
- Check if an accepting configuration can be reached
- Store only one computation path at a time (depth-first search)

Proving Savitch's Theorem

Simulating nondeterminism with more space:

- Use configuration graph of nondeterministic space-bounded TM
- Check if an accepting configuration can be reached
- Store only one computation path at a time (depth-first search)

This still requires exponential space. We want quadratic space!

What to do?

Proving Savitch's Theorem

Simulating nondeterminism with more space:

- Use configuration graph of nondeterministic space-bounded TM
- Check if an accepting configuration can be reached
- Store only one computation path at a time (depth-first search)

This still requires exponential space. We want quadratic space!

What to do?

Things we can do:

- Store one configuration:
 - one configuration requires $\log n + O(f(n))$ space
 - if $f(n) \geq \log n$, then this is $O(f(n))$ space
- Store $f(n)$ configurations (remember we have $f^2(n)$ space)
- Iterate over all configurations (one by one)

Proving Savitch's Theorem: Key Idea

To find out if we can reach an accepting configuration, we solve a slightly more general question:

YIELDABILITY

Input: TM configurations C_1 and C_2 , integer k

Problem: Can TM get from C_1 to C_2 in at most k steps?

Proving Savitch's Theorem: Key Idea

To find out if we can reach an accepting configuration, we solve a slightly more general question:

YIELDABILITY

Input: TM configurations C_1 and C_2 , integer k

Problem: Can TM get from C_1 to C_2 in at most k steps?

Approach: check if there is an intermediate configuration C' such that

- (1) C_1 can reach C' in $k/2$ steps and
- (2) C' can reach C_2 in $k/2$ steps

~> **Deterministic:** we can try all C' (iteration)

~> **Space-efficient:** we can reuse the same space for both steps

An Algorithm for Yieldability

```
01 CANYIELD( $C_1, C_2, k$ ) {  
02   if  $k = 1$  :  
03     return  $(C_1 = C_2)$  or  $(C_1 \vdash_{\mathcal{M}} C_2)$   
04   else if  $k > 1$  :  
05     for each configuration  $C$  of  $\mathcal{M}$  for input size  $n$  :  
06       if CANYIELD( $C_1, C, k/2$ ) and  
07         CANYIELD( $C, C_2, k/2$ ) :  
08         return true  
09   // eventually, if no success:  
10   return false  
11 }
```

- We only call CanYield only with k a power of 2, so $k/2 \in \mathbb{N}$

Space Requirement for the Algorithm

```
01 CANYIELD( $C_1, C_2, k$ ) {  
02   if  $k = 1$  :  
03     return ( $C_1 = C_2$ ) or ( $C_1 \vdash_M C_2$ )  
04   else if  $k > 1$  :  
05     for each configuration  $C$  of  $M$  for input size  $n$  :  
06       if CANYIELD( $C_1, C, k/2$ ) and  
07         CANYIELD( $C, C_2, k/2$ ) :  
08         return true  
09   // eventually, if no success:  
10   return false  
11 }
```

Space Requirement for the Algorithm

```
01 CANYIELD( $C_1, C_2, k$ ) {  
02   if  $k = 1$  :  
03     return ( $C_1 = C_2$ ) or ( $C_1 \vdash_M C_2$ )  
04   else if  $k > 1$  :  
05     for each configuration  $C$  of  $M$  for input size  $n$  :  
06       if CANYIELD( $C_1, C, k/2$ ) and  
07         CANYIELD( $C, C_2, k/2$ ) :  
08         return true  
09   // eventually, if no success:  
10   return false  
11 }
```

- During iteration (line 05), we store one C in $O(f(n))$

Space Requirement for the Algorithm

```
01 CANYIELD( $C_1, C_2, k$ ) {  
02   if  $k = 1$  :  
03     return ( $C_1 = C_2$ ) or ( $C_1 \vdash_M C_2$ )  
04   else if  $k > 1$  :  
05     for each configuration  $C$  of  $M$  for input size  $n$  :  
06       if CANYIELD( $C_1, C, k/2$ ) and  
07         CANYIELD( $C, C_2, k/2$ ) :  
08         return true  
09   // eventually, if no success:  
10   return false  
11 }
```

- During iteration (line 05), we store one C in $O(f(n))$
- Calls in lines 06 and 07 can reuse the same space

Space Requirement for the Algorithm

```
01 CANYIELD( $C_1, C_2, k$ ) {  
02   if  $k = 1$  :  
03     return ( $C_1 = C_2$ ) or ( $C_1 \vdash_M C_2$ )  
04   else if  $k > 1$  :  
05     for each configuration  $C$  of  $M$  for input size  $n$  :  
06       if CANYIELD( $C_1, C, k/2$ ) and  
07         CANYIELD( $C, C_2, k/2$ ) :  
08         return true  
09   // eventually, if no success:  
10   return false  
11 }
```

- During iteration (line 05), we store one C in $O(f(n))$
- Calls in lines 06 and 07 can reuse the same space
- Maximum depth of recursive call stack: $\log_2 k$

Space Requirement for the Algorithm

```
01 CANYIELD( $C_1, C_2, k$ ) {  
02   if  $k = 1$  :  
03     return  $(C_1 = C_2)$  or  $(C_1 \vdash_M C_2)$   
04   else if  $k > 1$  :  
05     for each configuration  $C$  of  $M$  for input size  $n$  :  
06       if CANYIELD( $C_1, C, k/2$ ) and  
07         CANYIELD( $C, C_2, k/2$ ) :  
08         return true  
09   // eventually, if no success:  
10   return false  
11 }
```

- During iteration (line 05), we store one C in $O(f(n))$
- Calls in lines 06 and 07 can reuse the same space
- Maximum depth of recursive call stack: $\log_2 k$

Overall space usage: $O(f(n) \cdot \log k)$

Simulating Nondeterministic Space-Bounded TMs

Input: TM $\mathcal{M}$ that runs in $\text{NSpace}(f(n))$; input word w of length n

Algorithm:

- Modify $\mathcal{M}$ to have a unique accepting configuration C_{accept} :
when accepting, erase tape and move head to the very left
- Select d such that $2^{df(n)} \geq |Q| \cdot n \cdot f(n) \cdot |\Gamma|^{f(n)}$
- Return $\text{CanYield}(C_{\text{start}}, C_{\text{accept}}, k)$ with $k = 2^{df(n)}$

Simulating Nondeterministic Space-Bounded TMs

Input: TM $\mathcal{M}$ that runs in $\text{NSpace}(f(n))$; input word w of length n

Algorithm:

- Modify $\mathcal{M}$ to have a unique accepting configuration C_{accept} :
when accepting, erase tape and move head to the very left
- Select d such that $2^{df(n)} \geq |Q| \cdot n \cdot f(n) \cdot |\Gamma|^{f(n)}$
- Return $\text{CanYield}(C_{\text{start}}, C_{\text{accept}}, k)$ with $k = 2^{df(n)}$

Space requirements:

CanYield runs in space

$$O(f(n) \cdot \log k) = O(f(n) \cdot \log 2^{df(n)}) = O(f(n) \cdot df(n)) = O(f^2(n))$$

Did We Really Do It?

Did We Really Do It?

“Select d such that $2^{df(n)} \geq |Q| \cdot n \cdot f(n) \cdot |\Gamma|^{f(n)}$ ”

How does the algorithm actually do this?

Did We Really Do It?

“Select d such that $2^{df(n)} \geq |Q| \cdot n \cdot f(n) \cdot |\Gamma|^{f(n)}$ ”

How does the algorithm actually do this?

- $f(n)$ was not part of the input!
- Even if we knew f , it might not be easy to compute!

Did We Really Do It?

“Select d such that $2^{df(n)} \geq |Q| \cdot n \cdot f(n) \cdot |\Gamma|^{f(n)}$ ”

How does the algorithm actually do this?

- $f(n)$ was not part of the input!
- Even if we knew f , it might not be easy to compute!

Solution: replace $f(n)$ by a parameter ℓ and probe its value

- (1) Start with $\ell = 1$
- (2) Check if $\mathcal{M}$ can reach any configuration with more than ℓ tape cells
(iterate over all configurations of size $\ell + 1$; use CanYield on each)
- (3) If yes, increase ℓ by 1; goto (2)
- (4) Run algorithm as before, with $f(n)$ replaced by ℓ

Therefore: we don't need to know f at all. This finishes the proof.

□

Polynomial Space

The Class PSpace

We defined PSpace as:

$$\text{PSpace} = \bigcup_{d \geq 1} \text{DSpace}(n^d)$$

and we observed that

$$P \subseteq NP \subseteq \text{PSpace} = \text{NPSpace} \subseteq \text{ExpTime}.$$

We can also define a corresponding notion of PSpace-hardness:

Definition 9.11:

- A language **H** is **PSpace-hard**, if $L \leq_p H$ for every language $L \in \text{PSpace}$.
- A language **C** is **PSpace-complete**, if **C** is PSpace-hard and $C \in \text{PSpace}$.

Quantified Boolean Formulae (QBF)

A **QBF** is a formula of the following form:

$$\mathcal{Q}_1 X_1 . \mathcal{Q}_2 X_2 . \dots \mathcal{Q}_\ell X_\ell . \varphi[X_1, \dots, X_\ell]$$

where $\mathcal{Q}_i \in \{\exists, \forall\}$ are quantifiers, X_i are propositional logic variables, and φ is a propositional logic formula with variables $X_1, \dots, X_\ell$ and constants $\top$ (true) and $\perp$ (false)

Semantics:

- Propositional formulae without variables (only constants $\top$ and $\perp$) are evaluated as usual
- $\exists X. \varphi[X]$ is true if either $\varphi[X/\top]$ or $\varphi[X/\perp]$ are true
- $\forall X. \varphi[X]$ is true if both $\varphi[X/\top]$ and $\varphi[X/\perp]$ are true

(where $\varphi[X/\top]$ is “ φ with X replaced by $\top$, and similar for $\perp$)

Deciding QBF Validity

TRUE QBF

Input: A quantified Boolean formula φ .

Problem: Is φ true (valid)?

Observation: We can assume that the quantified formula is in CNF or 3-CNF (same transformations possible as for propositional logic formulae)

Deciding QBF Validity

TRUE QBF

Input: A quantified Boolean formula φ .

Problem: Is φ true (valid)?

Observation: We can assume that the quantified formula is in CNF or 3-CNF (same transformations possible as for propositional logic formulae)

Consider a propositional logic formula φ with variables $X_1, \dots, X_\ell$:

Example 9.12: The QBF $\exists X_1 \dots \exists X_\ell. \varphi$ is true if and only if φ is satisfiable.

Example 9.13: The QBF $\forall X_1 \dots \forall X_\ell. \varphi$ is true if and only if φ is a tautology.

The Power of QBF

Theorem 9.14: **TRUE QBF** is PSpace-complete.

Proof:

(1) **TRUE QBF** $\in$ PSpace:

Give an algorithm that runs in polynomial space.

(2) **TRUE QBF** is PSpace-hard:

Proof by reduction from the word problem of any polynomially space-bounded TM.

Solving **TRUE QBF** in PSpace

```
01 TRUEQBF( $\varphi$ ) {  
02   if  $\varphi$  has no quantifiers :  
03     return “evaluation of  $\varphi$ ”  
04   else if  $\varphi = \exists X.\psi$  :  
05     return (TRUEQBF( $\psi[X/\top]$ ) OR TRUEQBF( $\psi[X/\perp]$ ))  
06   else if  $\varphi = \forall X.\psi$  :  
07     return (TRUEQBF( $\psi[X/\top]$ ) AND TRUEQBF( $\psi[X/\perp]$ ))  
08 }
```

- Evaluation in line 03 can be done in polynomial space
- Recursions in lines 05 and 07 can be executed one after the other, reusing space
- Maximum depth of recursion = number of variables (linear)
- Store one variable assignment per recursive call

$\leadsto$ polynomial space algorithm

PSpace-Hardness of **TRUE QBF**

Express TM computation in logic, similar to Cook-Levin

Given:

An arbitrary polynomially space-bounded NTM, that is:

- a polynomial p
- a p -space bounded 1-tape NTM $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}})$

Intended reduction

Given a word w , define a QBF $\varphi_{p, \mathcal{M}, w}$ such that

$\varphi_{p, \mathcal{M}, w}$ is true if and only if $\mathcal{M}$ accepts w in space $p(|w|)$.

Notes

- We show the reduction for NTMs, which is more than needed, but makes little difference in logic and allows us to reuse our previous formulae from Cook-Levin
- The proof actually shows many reductions, one for every polyspace NTM, showing PSpace-hardness from first principles

Review: Encoding Configurations

Use propositional variables for describing configurations:

Q_q for each $q \in Q$ means “ M is in state $q \in Q$ ”

P_i for each $0 \leq i < p(n)$ means “the head is at Position i ”

$S_{a,i}$ for each $a \in \Gamma$ and $0 \leq i < p(n)$ means “tape cell i contains Symbol a ”

Represent configuration $(q, p, a_0 \dots a_{p(n)})$

by assigning truth values to variables from the set

$$\overline{C} := \{Q_q, P_i, S_{a,i} \mid q \in Q, \quad a \in \Gamma, \quad 0 \leq i < p(n)\}$$

using the truth assignment β defined as

$$\beta(Q_s) := \begin{cases} 1 & s = q \\ 0 & s \neq q \end{cases} \quad \beta(P_i) := \begin{cases} 1 & i = p \\ 0 & i \neq p \end{cases} \quad \beta(S_{a,i}) := \begin{cases} 1 & a = a_i \\ 0 & a \neq a_i \end{cases}$$

Review: Validating Configurations

We define a formula $\text{Conf}(\overline{C})$ for a set of configuration variables

$$\overline{C} = \{Q_q, P_i, S_{a,i} \mid q \in Q, \quad a \in \Gamma, \quad 0 \leq i < p(n)\}$$

as follows:

$$\text{Conf}(\overline{C}) :=$$

“the assignment is a valid configuration”:

$$\bigvee_{q \in Q} (Q_q \wedge \bigwedge_{q' \neq q} \neg Q_{q'})$$

“TM in exactly one state $q \in Q$ ”

$$\wedge \bigvee_{p < p(n)} (P_p \wedge \bigwedge_{p' \neq p} \neg P_{p'})$$

“head in exactly one position $p < p(n)$ ”

$$\wedge \bigwedge_{0 \leq i < p(n)} \bigvee_{a \in \Gamma} (S_{a,i} \wedge \bigwedge_{b \neq a \in \Gamma} \neg S_{b,i})$$

“exactly one $a \in \Gamma$ in each cell”

Review: Validating Configurations

For an assignment β defined on variables in $\overline{C}$ define

$$\text{conf}(\overline{C}, \beta) := \left\{ (q, p, w_0 \dots w_{p(n)}) \mid \begin{array}{l} \beta(Q_q) = 1, \\ \beta(P_p) = 1, \\ \beta(S_{w_i, i}) = 1 \text{ for all } 0 \leq i < p(n) \end{array} \right\}$$

Note: β may be defined on other variables besides those in $\overline{C}$.

Lemma 9.15: If β satisfies $\text{Conf}(\overline{C})$ then $|\text{conf}(\overline{C}, \beta)| = 1$.

We can therefore write $\text{conf}(\overline{C}, \beta) = (q, p, w)$ to simplify notation.

Observations:

- $\text{conf}(\overline{C}, \beta)$ is a potential configuration of $\mathcal{M}$, but it may not be reachable from the start configuration of $\mathcal{M}$ on input w .
- Conversely, every configuration $(q, p, w_1 \dots w_{p(n)})$ induces a satisfying assignment β for which $\text{conf}(\overline{C}, \beta) = (q, p, w_1 \dots w_{p(n)})$.

Review: Transitions Between Configurations

Consider the following formula $\text{Next}(\overline{C}, \overline{C}')$ defined as

$$\text{Conf}(\overline{C}) \wedge \text{Conf}(\overline{C}') \wedge \text{NoChange}(\overline{C}, \overline{C}') \wedge \text{Change}(\overline{C}, \overline{C}').$$

$$\text{NoChange} := \bigvee_{0 \leq p < p(n)} \left(P_p \wedge \bigwedge_{i \neq p, a \in \Gamma} (S_{a,i} \rightarrow S'_{a,i}) \right)$$

$$\text{Change} := \bigvee_{0 \leq p < p(n)} \left(P_p \wedge \bigvee_{\substack{q \in Q \\ a \in \Gamma}} (Q_q \wedge S_{a,p} \wedge \bigvee_{(q', b, D) \in \delta(q, a)} (Q'_{q'} \wedge S'_{b,p} \wedge P'_{D(p)})) \right)$$

where $D(p)$ is the position reached by moving in direction D from p .

Lemma 9.16: For any assignment β defined on $\overline{C} \cup \overline{C}'$:

β satisfies $\text{Next}(\overline{C}, \overline{C}')$ if and only if $\text{conf}(\overline{C}, \beta) \vdash_{\mathcal{M}} \text{conf}(\overline{C}', \beta)$

Review: Start and End

Defined so far:

- $\text{Conf}(\bar{C})$: $\bar{C}$ describes a potential configuration
- $\text{Next}(\bar{C}, \bar{C}')$: $\text{conf}(\bar{C}, \beta) \vdash_{\mathcal{M}} \text{conf}(\bar{C}', \beta)$

Start configuration: Let $w = w_0 \cdots w_{n-1} \in \Sigma^*$ be the input word

$$\text{Start}_{\mathcal{M}, w}(\bar{C}) := \text{Conf}(\bar{C}) \wedge Q_{q_0} \wedge P_0 \wedge \bigwedge_{i=0}^{n-1} S_{w_i, i} \wedge \bigwedge_{i=n}^{p(n)-1} S_{\perp, i}$$

Then an assignment β satisfies $\text{Start}_{\mathcal{M}, w}(\bar{C})$ if and only if $\bar{C}$ represents the start configuration of $\mathcal{M}$ on input w .

Accepting stop configuration:

$$\text{Acc-Conf}(\bar{C}) := \text{Conf}(\bar{C}) \wedge Q_{q_{\text{accept}}}$$

Then an assignment β satisfies $\text{Acc-Conf}(\bar{C})$ if and only if $\bar{C}$ represents an accepting configuration of $\mathcal{M}$.

Simulating Polynomial Space Computations

For Cook-Levin, we used one set of configuration variables for every computing step:
polynomial time $\leadsto$ polynomially many variables

Problem: For polynomial space, we have $2^{O(p(n))}$ possible steps . . .

Simulating Polynomial Space Computations

For Cook-Levin, we used one set of configuration variables for every computing step:
polynomial time $\leadsto$ polynomially many variables

Problem: For polynomial space, we have $2^{O(p(n))}$ possible steps . . .

What would Savitch do?

Simulating Polynomial Space Computations

For Cook-Levin, we used one set of configuration variables for every computing step:
polynomial time $\leadsto$ polynomially many variables

Problem: For polynomial space, we have $2^{O(p(n))}$ possible steps . . .

What would Savitch do?

Define a formula $\text{CanYield}_i(\overline{C}_1, \overline{C}_2)$ to state that $\overline{C}_2$ is reachable from $\overline{C}_1$ in at most 2^i steps:

$$\text{CanYield}_0(\overline{C}_1, \overline{C}_2) := (\overline{C}_1 = \overline{C}_2) \vee \text{Next}(\overline{C}_1, \overline{C}_2)$$

$$\text{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2) := \exists \overline{C}. \text{Conf}(\overline{C}) \wedge \text{CanYield}_i(\overline{C}_1, \overline{C}) \wedge \text{CanYield}_i(\overline{C}, \overline{C}_2)$$

Simulating Polynomial Space Computations

For Cook-Levin, we used one set of configuration variables for every computing step:
polynomial time $\leadsto$ polynomially many variables

Problem: For polynomial space, we have $2^{O(p(n))}$ possible steps ...

What would Savitch do?

Define a formula $\text{CanYield}_i(\overline{C}_1, \overline{C}_2)$ to state that $\overline{C}_2$ is reachable from $\overline{C}_1$ in at most 2^i steps:

$$\text{CanYield}_0(\overline{C}_1, \overline{C}_2) := (\overline{C}_1 = \overline{C}_2) \vee \text{Next}(\overline{C}_1, \overline{C}_2)$$

$$\text{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2) := \exists \overline{C}. \text{Conf}(\overline{C}) \wedge \text{CanYield}_i(\overline{C}_1, \overline{C}) \wedge \text{CanYield}_i(\overline{C}, \overline{C}_2)$$

But what is $\overline{C}_1 = \overline{C}_2$ supposed to mean here?

Simulating Polynomial Space Computations

For Cook-Levin, we used one set of configuration variables for every computing step:
polynomial time $\leadsto$ polynomially many variables

Problem: For polynomial space, we have $2^{O(p(n))}$ possible steps ...

What would Savitch do?

Define a formula $\text{CanYield}_i(\overline{C}_1, \overline{C}_2)$ to state that $\overline{C}_2$ is reachable from $\overline{C}_1$ in at most 2^i steps:

$$\text{CanYield}_0(\overline{C}_1, \overline{C}_2) := (\overline{C}_1 = \overline{C}_2) \vee \text{Next}(\overline{C}_1, \overline{C}_2)$$

$$\text{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2) := \exists \overline{C}. \text{Conf}(\overline{C}) \wedge \text{CanYield}_i(\overline{C}_1, \overline{C}) \wedge \text{CanYield}_i(\overline{C}, \overline{C}_2)$$

But what is $\overline{C}_1 = \overline{C}_2$ supposed to mean here? It is short for:

$$\bigwedge_{q \in Q} Q_q^1 \leftrightarrow Q_q^2 \wedge \bigwedge_{0 \leq i < p(n)} P_i^1 \leftrightarrow P_i^2 \wedge \bigwedge_{a \in \Gamma, 0 \leq i < p(n)} S_{a,i}^1 \leftrightarrow S_{a,i}^2$$

Putting Everything Together

We define the formula $\varphi_{p,\mathcal{M},w}$ as follows:

$$\varphi_{p,\mathcal{M},w} := \exists \bar{C}_1. \exists \bar{C}_2. \text{Start}_{\mathcal{M},w}(\bar{C}_1) \wedge \text{Acc-Conf}(\bar{C}_2) \wedge \text{CanYield}_{dp(n)}(\bar{C}_1, \bar{C}_2)$$

where we select d to be the least number such that $\mathcal{M}$ has less than $2^{dp(n)}$ configurations in space $p(n)$.

Lemma 9.17: $\varphi_{p,\mathcal{M},w}$ is satisfiable if and only if $\mathcal{M}$ accepts w in space $p(|w|)$.

Did we do it?

Note: we used only existential quantifiers when defining $\varphi_{p,\mathcal{M},w}$:

$$\text{CanYield}_0(\overline{C}_1, \overline{C}_2) := (\overline{C}_1 = \overline{C}_2) \vee \text{Next}(\overline{C}_1, \overline{C}_2)$$

$$\text{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2) := \exists \overline{C}. \text{Conf}(\overline{C}) \wedge \text{CanYield}_i(\overline{C}_1, \overline{C}) \wedge \text{CanYield}_i(\overline{C}, \overline{C}_2)$$

$$\varphi_{p,\mathcal{M},w} := \exists \overline{C}_1. \exists \overline{C}_2. \text{Start}_{\mathcal{M},w}(\overline{C}_1) \wedge \text{Acc-Conf}(\overline{C}_2) \wedge \text{CanYield}_{dp(n)}(\overline{C}_1, \overline{C}_2)$$

Did we do it?

Note: we used only existential quantifiers when defining $\varphi_{p,\mathcal{M},w}$:

$$\text{CanYield}_0(\overline{C}_1, \overline{C}_2) := (\overline{C}_1 = \overline{C}_2) \vee \text{Next}(\overline{C}_1, \overline{C}_2)$$

$$\text{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2) := \exists \overline{C}. \text{Conf}(\overline{C}) \wedge \text{CanYield}_i(\overline{C}_1, \overline{C}) \wedge \text{CanYield}_i(\overline{C}, \overline{C}_2)$$

$$\varphi_{p,\mathcal{M},w} := \exists \overline{C}_1. \exists \overline{C}_2. \text{Start}_{\mathcal{M},w}(\overline{C}_1) \wedge \text{Acc-Conf}(\overline{C}_2) \wedge \text{CanYield}_{dp(n)}(\overline{C}_1, \overline{C}_2)$$

Now that's quite interesting . . .

Did we do it?

Note: we used only existential quantifiers when defining $\varphi_{p,\mathcal{M},w}$:

$$\text{CanYield}_0(\overline{C}_1, \overline{C}_2) := (\overline{C}_1 = \overline{C}_2) \vee \text{Next}(\overline{C}_1, \overline{C}_2)$$

$$\text{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2) := \exists \overline{C}. \text{Conf}(\overline{C}) \wedge \text{CanYield}_i(\overline{C}_1, \overline{C}) \wedge \text{CanYield}_i(\overline{C}, \overline{C}_2)$$

$$\varphi_{p,\mathcal{M},w} := \exists \overline{C}_1. \exists \overline{C}_2. \text{Start}_{\mathcal{M},w}(\overline{C}_1) \wedge \text{Acc-Conf}(\overline{C}_2) \wedge \text{CanYield}_{dp(n)}(\overline{C}_1, \overline{C}_2)$$

Now that's quite interesting . . .

- With only (non-negated) $\exists$ quantifiers, **TRUE QBF** coincides with **SAT**

Did we do it?

Note: we used only existential quantifiers when defining $\varphi_{p,\mathcal{M},w}$:

$$\text{CanYield}_0(\overline{C}_1, \overline{C}_2) := (\overline{C}_1 = \overline{C}_2) \vee \text{Next}(\overline{C}_1, \overline{C}_2)$$

$$\text{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2) := \exists \overline{C}. \text{Conf}(\overline{C}) \wedge \text{CanYield}_i(\overline{C}_1, \overline{C}) \wedge \text{CanYield}_i(\overline{C}, \overline{C}_2)$$

$$\varphi_{p,\mathcal{M},w} := \exists \overline{C}_1. \exists \overline{C}_2. \text{Start}_{\mathcal{M},w}(\overline{C}_1) \wedge \text{Acc-Conf}(\overline{C}_2) \wedge \text{CanYield}_{dp(n)}(\overline{C}_1, \overline{C}_2)$$

Now that's quite interesting . . .

- With only (non-negated) $\exists$ quantifiers, **TRUE QBF** coincides with **SAT**
- **SAT** is in NP

Did we do it?

Note: we used only existential quantifiers when defining $\varphi_{p,\mathcal{M},w}$:

$$\text{CanYield}_0(\overline{C}_1, \overline{C}_2) := (\overline{C}_1 = \overline{C}_2) \vee \text{Next}(\overline{C}_1, \overline{C}_2)$$

$$\text{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2) := \exists \overline{C}. \text{Conf}(\overline{C}) \wedge \text{CanYield}_i(\overline{C}_1, \overline{C}) \wedge \text{CanYield}_i(\overline{C}, \overline{C}_2)$$

$$\varphi_{p,\mathcal{M},w} := \exists \overline{C}_1. \exists \overline{C}_2. \text{Start}_{\mathcal{M},w}(\overline{C}_1) \wedge \text{Acc-Conf}(\overline{C}_2) \wedge \text{CanYield}_{dp(n)}(\overline{C}_1, \overline{C}_2)$$

Now that's quite interesting . . .

- With only (non-negated) $\exists$ quantifiers, **TRUE QBF** coincides with **SAT**
- **SAT** is in NP
- So we showed that the word problem for PSpace NTMs to be in NP

Did we do it?

Note: we used only existential quantifiers when defining $\varphi_{P, \mathcal{M}, w}$:

$$\text{CanYield}_0(\overline{C}_1, \overline{C}_2) := (\overline{C}_1 = \overline{C}_2) \vee \text{Next}(\overline{C}_1, \overline{C}_2)$$

$$\text{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2) := \exists \overline{C}. \text{Conf}(\overline{C}) \wedge \text{CanYield}_i(\overline{C}_1, \overline{C}) \wedge \text{CanYield}_i(\overline{C}, \overline{C}_2)$$

$$\varphi_{P, \mathcal{M}, w} := \exists \overline{C}_1. \exists \overline{C}_2. \text{Start}_{\mathcal{M}, w}(\overline{C}_1) \wedge \text{Acc-Conf}(\overline{C}_2) \wedge \text{CanYield}_{dp(n)}(\overline{C}_1, \overline{C}_2)$$

Now that's quite interesting . . .

- With only (non-negated) $\exists$ quantifiers, **TRUE QBF** coincides with **SAT**
- **SAT** is in NP
- So we showed that the word problem for PSpace NTMs to be in NP

So we found that **NP = PSpace!**

Did we do it?

Note: we used only existential quantifiers when defining $\varphi_{P, \mathcal{M}, w}$:

$$\text{CanYield}_0(\overline{C}_1, \overline{C}_2) := (\overline{C}_1 = \overline{C}_2) \vee \text{Next}(\overline{C}_1, \overline{C}_2)$$

$$\text{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2) := \exists \overline{C}. \text{Conf}(\overline{C}) \wedge \text{CanYield}_i(\overline{C}_1, \overline{C}) \wedge \text{CanYield}_i(\overline{C}, \overline{C}_2)$$

$$\varphi_{P, \mathcal{M}, w} := \exists \overline{C}_1. \exists \overline{C}_2. \text{Start}_{\mathcal{M}, w}(\overline{C}_1) \wedge \text{Acc-Conf}(\overline{C}_2) \wedge \text{CanYield}_{dp(n)}(\overline{C}_1, \overline{C}_2)$$

Now that's quite interesting . . .

- With only (non-negated) $\exists$ quantifiers, **TRUE QBF** coincides with **SAT**
- **SAT** is in NP
- So we showed that the word problem for PSpace NTMs to be in NP

So we found that **NP = PSpace!**

Strangely, most textbooks claim that this is not known to be true . . .

Are we up for the next Turing Award, or did we make a **mistake?**

Size

How big is $\varphi_{p,\mathcal{M},w}$?

$$\text{CanYield}_0(\overline{C}_1, \overline{C}_2) := (\overline{C}_1 = \overline{C}_2) \vee \text{Next}(\overline{C}_1, \overline{C}_2)$$

$$\text{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2) := \exists \overline{C}. \text{Conf}(\overline{C}) \wedge \text{CanYield}_i(\overline{C}_1, \overline{C}) \wedge \text{CanYield}_i(\overline{C}, \overline{C}_2)$$

$$\varphi_{p,\mathcal{M},w} := \exists \overline{C}_1. \exists \overline{C}_2. \text{Start}_{\mathcal{M},w}(\overline{C}_1) \wedge \text{Acc-Conf}(\overline{C}_2) \wedge \text{CanYield}_{dp(n)}(\overline{C}_1, \overline{C}_2)$$

Size

How big is $\varphi_{p, \mathcal{M}, w}$?

$$\text{CanYield}_0(\overline{C}_1, \overline{C}_2) := (\overline{C}_1 = \overline{C}_2) \vee \text{Next}(\overline{C}_1, \overline{C}_2)$$

$$\text{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2) := \exists \overline{C}. \text{Conf}(\overline{C}) \wedge \text{CanYield}_i(\overline{C}_1, \overline{C}) \wedge \text{CanYield}_i(\overline{C}, \overline{C}_2)$$

$$\varphi_{p, \mathcal{M}, w} := \exists \overline{C}_1. \exists \overline{C}_2. \text{Start}_{\mathcal{M}, w}(\overline{C}_1) \wedge \text{Acc-Conf}(\overline{C}_2) \wedge \text{CanYield}_{dp(n)}(\overline{C}_1, \overline{C}_2)$$

Size of CanYield_{i+1} is more than twice the size of CanYield_i

$\leadsto$ Size of $\varphi_{p, \mathcal{M}, w}$ is in $2^{O(p(n))}$. Oops.

Size

How big is $\varphi_{p, \mathcal{M}, w}$?

$$\text{CanYield}_0(\overline{C}_1, \overline{C}_2) := (\overline{C}_1 = \overline{C}_2) \vee \text{Next}(\overline{C}_1, \overline{C}_2)$$

$$\text{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2) := \exists \overline{C}. \text{Conf}(\overline{C}) \wedge \text{CanYield}_i(\overline{C}_1, \overline{C}) \wedge \text{CanYield}_i(\overline{C}, \overline{C}_2)$$

$$\varphi_{p, \mathcal{M}, w} := \exists \overline{C}_1. \exists \overline{C}_2. \text{Start}_{\mathcal{M}, w}(\overline{C}_1) \wedge \text{Acc-Conf}(\overline{C}_2) \wedge \text{CanYield}_{dp(n)}(\overline{C}_1, \overline{C}_2)$$

Size of CanYield_{i+1} is more than twice the size of CanYield_i

$\leadsto$ Size of $\varphi_{p, \mathcal{M}, w}$ is in $2^{O(p(n))}$. Oops.

A correct reduction: We redefine CanYield by setting

$$\text{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2) :=$$

$$\exists \overline{C}. \text{Conf}(\overline{C}) \wedge$$

$$\forall \overline{Z}_1. \forall \overline{Z}_2. (((\overline{Z}_1 = \overline{C}_1 \wedge \overline{Z}_2 = \overline{C}) \vee (\overline{Z}_1 = \overline{C} \wedge \overline{Z}_2 = \overline{C}_2)) \rightarrow \text{CanYield}_i(\overline{Z}_1, \overline{Z}_2))$$

Size

Let's analyse the size more carefully this time:

$$\begin{aligned}\text{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2) &:= \\ \exists \overline{C}. \text{Conf}(\overline{C}) \wedge \\ \forall \overline{Z}_1. \forall \overline{Z}_2. (((\overline{Z}_1 = \overline{C}_1 \wedge \overline{Z}_2 = \overline{C}) \vee (\overline{Z}_1 = \overline{C} \wedge \overline{Z}_2 = \overline{C}_2)) \rightarrow \text{CanYield}_i(\overline{Z}_1, \overline{Z}_2))\end{aligned}$$

- $\text{CanYield}_{i+1}(\overline{C}_1, \overline{C}_2)$ extends $\text{CanYield}_i(\overline{C}_1, \overline{C}_2)$ by parts that are linear in the size of configurations $\leadsto$ growth in $O(p(n))$
- Maximum index i used in $\varphi_{p, \mathcal{M}, w}$ is $dp(n)$, that is in $O(p(n))$
- Therefore: $\varphi_{p, \mathcal{M}, w}$ has size $O(p^2(n))$ – and thus can be computed in polynomial time

Exercise:

Why can we just use $dp(n)$ in the reduction? Don't we have to compute it somehow? Maybe even in polynomial time?

Summary: Relationships of Space and Time

Summing up, we get the following relations:

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSpace = NPSPACE \subseteq ExpTime \subseteq NExpTime$$

We also noted $P \subseteq coNP \subseteq PSpace$.

Open questions:

- Is Savitch's Theorem tight?
- Are there any interesting problems in these space classes?
- We have $PSpace = NPSPACE = coNPSPACE$.
But what about L , NL , and $coNL$?

Summary: Relationships of Space and Time

Summing up, we get the following relations:

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSpace = NPSPACE \subseteq ExpTime \subseteq NExpTime$$

We also noted $P \subseteq coNP \subseteq PSpace$.

Open questions:

- Is Savitch's Theorem tight?
- Are there any interesting problems in these space classes?
- We have $PSpace = NPSPACE = coNPSPACE$.
But what about L , NL , and $coNL$?

↪ the first: **nobody knows** (YCTBF); the others: see upcoming lectures