# Automata Theory :: Complexity

Jörg Endrullis

Vrije Universiteit Amsterdam

# Big O Notation

Let $f, g : \mathbb{N} \to \mathbb{R}_{>0}$. Then

$$f \in O(g) \quad \iff \quad \exists C > 0.\ \exists n_0.\ f(n) \leq C \cdot g(n) \text{ for all } n \geq n_0$$

# Big O Notation

Let $f, g : \mathbb{N} \to \mathbb{R}_{>0}$. Then

$$f \in O(g) \iff \exists C > 0.\ \exists n_0.\ f(n) \leq C \cdot g(n) \text{ for all } n \geq n_0$$

$$\begin{aligned}
n^a &\in O(n^b) && \text{for all } 0 < a \leq b \\
c_a n^a + c_{a-1} n^{a-1} + \cdots + c_0 &\in O(n^a) && \text{for all } a > 0 \\
n^a &\in O(b^n) && \text{for all } a > 0 \text{ and } b > 1 \\
\log_a n &\in O(n^b) && \text{for all } a, b > 0 \\
\log_a n &\in O(\log_b n) && \text{for all } a, b > 0
\end{aligned}$$

# Big O Notation

Let $f, g : \mathbb{N} \to \mathbb{R}_{>0}$. Then

$$f \in O(g) \quad \Longleftrightarrow \quad \exists C > 0.\ \exists n_0.\ f(n) \leq C \cdot g(n) \text{ for all } n \geq n_0$$

$$\begin{aligned}
n^a &\in O(n^b) && \text{for all } 0 < a \leq b \\
c_a n^a + c_{a-1} n^{a-1} + \cdots + c_0 &\in O(n^a) && \text{for all } a > 0 \\
n^a &\in O(b^n) && \text{for all } a > 0 \text{ and } b > 1 \\
\log_a n &\in O(n^b) && \text{for all } a, b > 0 \\
\log_a n &\in O(\log_b n) && \text{for all } a, b > 0
\end{aligned}$$

By definition $\log_a a^n = n$.

# Big O Notation

Let $f, g : \mathbb{N} \to \mathbb{R}_{>0}$. Then

$$f \in O(g) \quad\iff\quad \exists C > 0.\ \exists n_0.\ f(n) \leq C \cdot g(n) \text{ for all } n \geq n_0$$

$$\begin{aligned}
n^a &\in O(n^b) && \text{for all } 0 < a \leq b \\
c_a n^a + c_{a-1} n^{a-1} + \cdots + c_0 &\in O(n^a) && \text{for all } a > 0 \\
n^a &\in O(b^n) && \text{for all } a > 0 \text{ and } b > 1 \\
\log_a n &\in O(n^b) && \text{for all } a, b > 0 \\
\log_a n &\in O(\log_b n) && \text{for all } a, b > 0
\end{aligned}$$

By definition $\log_a a^n = n$. This implies $a^{\log_a n} = n$

# Big O Notation

Let $f, g : \mathbb{N} \to \mathbb{R}_{>0}$. Then

$$f \in O(g) \quad \iff \quad \exists C > 0.\ \exists n_0.\ f(n) \leq C \cdot g(n) \text{ for all } n \geq n_0$$

$$\begin{aligned}
n^a &\in O(n^b) && \text{for all } 0 < a \leq b \\
c_a n^a + c_{a-1} n^{a-1} + \cdots + c_0 &\in O(n^a) && \text{for all } a > 0 \\
n^a &\in O(b^n) && \text{for all } a > 0 \text{ and } b > 1 \\
\log_a n &\in O(n^b) && \text{for all } a, b > 0 \\
\log_a n &\in O(\log_b n) && \text{for all } a, b > 0
\end{aligned}$$

By definition $\log_a a^n = n$. This implies $a^{\log_a n} = n$, and hence

$$a^{\log_a b \cdot \log_b n} = \left(a^{\log_a b}\right)^{\log_b n} = b^{\log_b n} = n$$

Hence $\log_a b \cdot \log_b n = \log_a n$.

# Time Complexity: P and NP

# Time Complexity

Let $f, g : \mathbb{N} \to \mathbb{N}$.

A nondeterministic Turing machine $M$

**runs in time $f$**

if for every input $w$, every computation of $M$ reaches a halting state after at most $f(|w|)$ steps.

# Time Complexity

Let $f, g : \mathbb{N} \to \mathbb{N}$.

A nondeterministic Turing machine $M$

**runs in time** $f$

if for every input $w$, every computation of $M$ reaches a halting state after at most $f(|w|)$ steps.

The function $f$ gives an upper bound on the number of computation steps in terms of the length of the input word.

# Time Complexity

Let $f, g : \mathbb{N} \to \mathbb{N}$.

A nondeterministic Turing machine $M$

**runs in time** $f$

if for every input $w$, every computation of $M$ reaches a halting state after at most $f(|w|)$ steps.

The function $f$ gives an upper bound on the number of computation steps in terms of the length of the input word.

A Turing machine $M$ has

**time complexity** $O(g)$

if there exists $f \in O(g)$ such that $M$ runs in time $f$.

# Complexity Classes P and NP

A nondeterministic Turing machine $M$ is **polynomial time** if $M$ runs in time $p$ for some polynomial $p$.

Equivalently, $M$ has time complexity $O(n^k)$ for some $k$.

# Complexity Classes P and NP

A nondeterministic Turing machine $M$ is **polynomial time** if $M$ runs in time $p$ for some polynomial $p$.

Equivalently, $M$ has time complexity $O(n^k)$ for some $k$.

**NP** is the class of languages accepted by nondeterministic polynomial time Turing machines:

$$\mathbf{NP} = \{ L(M) \mid M \text{ is nondeterministic polynomial time TM} \}$$

# Complexity Classes P and NP

A nondeterministic Turing machine $M$ is **polynomial time** if $M$ runs in time $p$ for some polynomial $p$.

Equivalently, $M$ has time complexity $O(n^k)$ for some $k$.

**NP** is the class of languages accepted by nondeterministic polynomial time Turing machines:

$$\mathbf{NP} = \{\, L(M) \mid M \text{ is nondeterministic polynomial time TM} \,\}$$

**P** is the class of languages accepted by deterministic polynomial time Turing machines:

$$\mathbf{P} = \{\, L(M) \mid M \text{ is deterministic polynomial time TM} \,\}$$

# Complexity Classes P and NP

A nondeterministic Turing machine $M$ is **polynomial time** if $M$ runs in time $p$ for some polynomial $p$.

Equivalently, $M$ has time complexity $O(n^k)$ for some $k$.

**NP** is the class of languages accepted by nondeterministic polynomial time Turing machines:

$$\mathbf{NP} = \{\, L(M) \mid M \text{ is nondeterministic polynomial time TM} \,\}$$

**P** is the class of languages accepted by deterministic polynomial time Turing machines:

$$\mathbf{P} = \{\, L(M) \mid M \text{ is deterministic polynomial time TM} \,\}$$

Clearly P $\subseteq$ NP, but it is unknown whether P $=$ NP.

# Problems in NP

Recall, that the language corresponding to a decision problem consists of words representing instances of the problem for which the answer is **yes**.

# Problems in NP

Recall, that the language corresponding to a decision problem consists of words representing instances of the problem for which the answer is **yes**.

Intuitively a problem is in NP if:

- every instance has a finite set of possible solutions,
- correctness of a solution can be checked in polynomial time

# Problems in NP

Recall, that the language corresponding to a decision problem consists of words representing instances of the problem for which the answer is **yes**.

Intuitively a problem is in NP if:

- every instance has a finite set of possible solutions,
- correctness of a solution can be checked in polynomial time

The question whether the **travelling salesman problem** has a solution of length $\leq k$ is in NP.

# Problems in NP

Recall, that the language corresponding to a decision problem consists of words representing instances of the problem for which the answer is **yes**.

Intuitively a problem is in NP if:

- every instance has a finite set of possible solutions,
- correctness of a solution can be checked in polynomial time

The question whether the **travelling salesman problem** has a solution of length $\leq k$ is in NP.

**Satisfiability in propositional logic** is in NP.

# Problems in NP

Recall, that the language corresponding to a decision problem consists of words representing instances of the problem for which the answer is **yes**.

Intuitively a problem is in NP if:

- every instance has a finite set of possible solutions,
- correctness of a solution can be checked in polynomial time

The question whether the **travelling salesman problem** has a solution of length $\leq k$ is in NP.


**Satisfiability in propositional logic** is in NP.

The questions if a number is **not prime** is in NP.

# Problems in NP

Recall, that the language corresponding to a decision problem consists of words representing instances of the problem for which the answer is **yes**.

Intuitively a problem is in NP if:

- every instance has a finite set of possible solutions,
- correctness of a solution can be checked in polynomial time

The question whether the **travelling salesman problem** has a solution of length $\leq k$ is in NP.


**Satisfiability in propositional logic** is in NP.

The questions if a number is **not prime** is in NP.

Surprisingly, last question in P. (Agrawal, Kayal, Saxena, 2002)

# Satisfiability in Propositional Logic

A formula of propositional logic consists of

| | | |
|---|---|---|
| `true` | conjunction $\wedge$ | variables |
| `false` | disjunction $\vee$ | negation $\neg$ |

A formula of propositional logic $\phi$ is **satisfiable** if there exists an assignment of `true` and `false` to the variables such that $\phi$ evaluates to `true`.

# Satisfiability in Propositional Logic

A formula of propositional logic consists of

| | | |
|---|---|---|
| `true` | conjunction $\wedge$ | variables |
| `false` | disjunction $\vee$ | negation $\neg$ |

A formula of propositional logic $\phi$ is **satisfiable** if there exists an assignment of `true` and `false` to the variables such that $\phi$ evaluates to `true`.

**Theorem**

Satisfiability of formulas of propositional logic is in NP.

# Satisfiability in Propositional Logic

A formula of propositional logic consists of

| | | |
|---|---|---|
| `true` | conjunction $\wedge$ | variables |
| `false` | disjunction $\vee$ | negation $\neg$ |

A formula of propositional logic $\phi$ is **satisfiable** if there exists an assignment of `true` and `false` to the variables such that $\phi$ evaluates to `true`.

### Theorem

Satisfiability of formulas of propositional logic is in NP.

### Proof.

We can construct a nondeterministic Turing machine that

- guesses an assignment of `true` and `false` to the variables,
- evaluates the formula (in polynomial time), and

accepts if the evaluation is `true`. □

# NP-completeness

# NP-completeness

Let $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ be decision problems (languages).

Then $L_1$ is **polynomial-time reducible** to $L_2$ if there exists a **polynomial-time computable** function $f : \Sigma_1^* \to \Sigma_2^*$ such that:

$$x \in L_1 \iff f(x) \in L_2$$

# NP-completeness

Let $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ be decision problems (languages).

Then $L_1$ is **polynomial-time reducible** to $L_2$ if there exists a **polynomial-time computable** function $f : \Sigma_1^* \to \Sigma_2^*$ such that:

$$x \in L_1 \iff f(x) \in L_2$$

To decide if $x \in L_1$, we can compute $f(x)$ and check $f(x) \in L_2$.

So the problem $L_1$ is reduced to the problem $L_2$.

# NP-completeness

Let $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ be decision problems (languages).

Then $L_1$ is **polynomial-time reducible** to $L_2$ if there exists a **polynomial-time computable** function $f : \Sigma_1^* \to \Sigma_2^*$ such that:

$$x \in L_1 \iff f(x) \in L_2$$

To decide if $x \in L_1$, we can compute $f(x)$ and check $f(x) \in L_2$.

So the problem $L_1$ is reduced to the problem $L_2$.

Let $f : \Sigma_1^* \to \Sigma_2^*$ and $g : \Sigma_2^* \to \Sigma_3^*$ be polynomial-time reductions.
The composition $g \circ f : \Sigma_1^* \to \Sigma_3^*$ is a polynomial-time reduction.

# NP-completeness

Let $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ be decision problems (languages).

Then $L_1$ is **polynomial-time reducible** to $L_2$ if there exists a **polynomial-time computable** function $f : \Sigma_1^* \to \Sigma_2^*$ such that:

$$x \in L_1 \iff f(x) \in L_2$$

To decide if $x \in L_1$, we can compute $f(x)$ and check $f(x) \in L_2$.

So the problem $L_1$ is reduced to the problem $L_2$.

Let $f : \Sigma_1^* \to \Sigma_2^*$ and $g : \Sigma_2^* \to \Sigma_3^*$ be polynomial-time reductions. The composition $g \circ f : \Sigma_1^* \to \Sigma_3^*$ is a polynomial-time reduction.

## NP-completeness

A language $L \in \mathrm{NP}$ is **NP-complete** if every language in NP is polynomial time reducible to $L$.

# Examples of NP-complete Problems

The question whether the **travelling salesman problem** has a solution of length $\leq k$ is NP-complete.

# Examples of NP-complete Problems

The question whether the **travelling salesman problem** has a solution of length $\leq k$ is NP-complete.

**Satisfiability** for formulas of propositional logic is NP-complete.

# Examples of NP-complete Problems

The question whether the **travelling salesman problem** has a solution of length $\leq k$ is NP-complete.

**Satisfiability** for formulas of propositional logic is NP-complete.

The question whether a graph contains a **Hamiltonian cycle** (a cycle that visits each node exactly once) is NP-complete.

# Examples of NP-complete Problems

The question whether the **travelling salesman problem** has a solution of length $\leq k$ is NP-complete.

**Satisfiability** for formulas of propositional logic is NP-complete.

The question whether a graph contains a **Hamiltonian cycle** (a cycle that visits each node exactly once) is NP-complete.

The **bounded tiling problem** is NP-complete.

# Examples of NP-complete Problems

The question whether the **travelling salesman problem** has a solution of length $\leq k$ is NP-complete.

**Satisfiability** for formulas of propositional logic is NP-complete.

The question whether a graph contains a **Hamiltonian cycle** (a cycle that visits each node exactly once) is NP-complete.

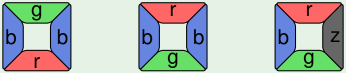
The **bounded tiling problem** is NP-complete.

…and many more questions

# Bounded Tiling Problem

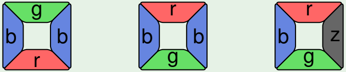
# Bounded Tiling Problem

Given a finite collection of **types** of $1 \times 1$ **tiles** with a **colour** on each side. (There are infinitely many tiles of each type.)



# Bounded Tiling Problem

Given a finite collection of **types** of $1 \times 1$ **tiles** with a **colour** on each side. (There are infinitely many tiles of each type.)



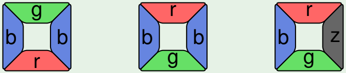

**Bonded tiling problem**: the input is $n \in \mathbb{N}$, a finite collection of types of tiles, the first row of $n$ tiles.

Is it possible to tile an $n \times n$ field (with the given first row)?

When connecting tiles, the touching side must have the same colour. Tiles must not be rotated.

# Bounded Tiling Problem

Given a finite collection of **types** of $1 \times 1$ **tiles** with a **colour** on each side. (There are infinitely many tiles of each type.)

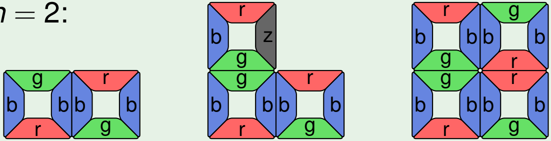



**Bonded tiling problem**: the input is $n \in \mathbb{N}$, a finite collection of types of tiles, the first row of $n$ tiles.

Is it possible to tile an $n \times n$ field (with the given first row)?

When connecting tiles, the touching side must have the same colour. Tiles must not be rotated.

Example $n = 2$:



first row incomplete tiling correct tiling

# Bounded Tiling Problem is NP-complete

**Theorem**

The bounded tiling problem is NP-complete.

# Bounded Tiling Problem is NP-complete

**Theorem**

The bounded tiling problem is NP-complete.

**Proof**

**First**, we argue that the bounded tiling problem is in NP.

# Bounded Tiling Problem is NP-complete

## Theorem

The bounded tiling problem is NP-complete.

## Proof

**First**, we argue that the bounded tiling problem is in NP.

We can construct a nondeterministic Turing machien that

- guesses an $n \times n$ tiling, and
- afterwards checks whether the solution is correct.

Both steps can be done in polynomial time.

# Bounded Tiling Problem is NP-complete

## Theorem

The bounded tiling problem is NP-complete.

## Proof

**First**, we argue that the bounded tiling problem is in NP.

We can construct a nondeterministic Turing machien that

- guesses an $n \times n$ tiling, and
- afterwards checks whether the solution is correct.

Both steps can be done in polynomial time.

**Second**, we show NP-completeness.

# Bounded Tiling Problem is NP-complete

## Theorem

The bounded tiling problem is NP-complete.

## Proof

**First**, we argue that the bounded tiling problem is in NP.

We can construct a nondeterministic Turing machien that

- guesses an $n \times n$ tiling, and
- afterwards checks whether the solution is correct.

Both steps can be done in polynomial time.

**Second**, we show NP-completeness.

Let $M$ be a nondeterministic polynomial-time Turing machine.

# Bounded Tiling Problem is NP-complete

## Theorem

The bounded tiling problem is NP-complete.

## Proof

**First**, we argue that the bounded tiling problem is in NP.

We can construct a nondeterministic Turing machien that

- guesses an $n \times n$ tiling, and
- afterwards checks whether the solution is correct.

Both steps can be done in polynomial time.

**Second**, we show NP-completeness.

Let $M$ be a nondeterministic polynomial-time Turing machine.
Then $M$ has running time $p(k)$ for some polynomial $p(k)$.

# Bounded Tiling Problem is NP-complete

### Theorem

The bounded tiling problem is NP-complete.

### Proof

**First**, we argue that the bounded tiling problem is in NP.

We can construct a nondeterministic Turing machien that

- guesses an $n \times n$ tiling, and
- afterwards checks whether the solution is correct.

Both steps can be done in polynomial time.

**Second**, we show NP-completeness.

Let $M$ be a nondeterministic polynomial-time Turing machine.
Then $M$ has running time $p(k)$ for some polynomial $p(k)$.

We give a polynomial-time reduction of $x \in L(M)\,?$ to the bounded tiling problem.

# Bounded Tiling Problem is NP-complete

## Theorem

The bounded tiling problem is NP-complete.

## Proof

**First**, we argue that the bounded tiling problem is in NP.

We can construct a nondeterministic Turing machien that

- guesses an $n \times n$ tiling, and
- afterwards checks whether the solution is correct.

Both steps can be done in polynomial time.

**Second**, we show NP-completeness.

Let $M$ be a nondeterministic polynomial-time Turing machine.
Then $M$ has running time $p(k)$ for some polynomial $p(k)$.

We give a polynomial-time reduction of $x \in L(M)\,?$ to the bounded tiling problem.

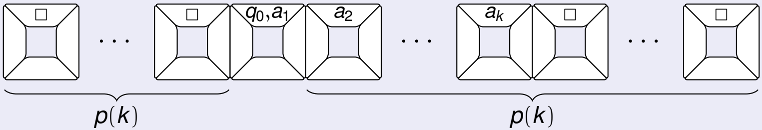
continued on the next slide. . .

# Bounded Tiling Problem is NP-complete

## Proof continued... (the starting row)

For input word $x = a_1 \cdots a_k$ we choose $n = 2p(k) + 1$.
(Assume $p(k) \geq k$, otherwise make it so.)

As first row we choose:

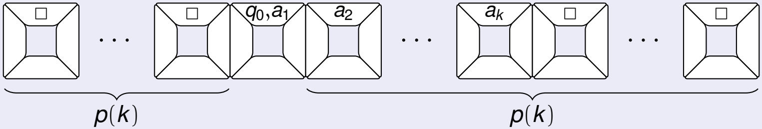


# Bounded Tiling Problem is NP-complete

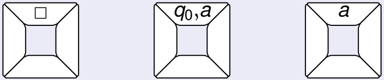
## Proof continued... (the starting row)

For input word $x = a_1 \cdots a_k$ we choose $n = 2p(k) + 1$.
(Assume $p(k) \geq k$, otherwise make it so.)

As first row we choose:




Tiles for building the first row (for every $a \in \Sigma$):

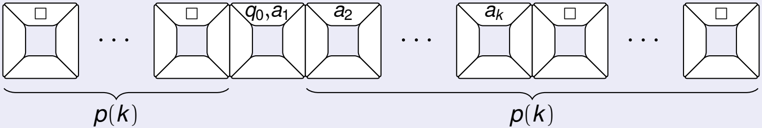


# Bounded Tiling Problem is NP-complete

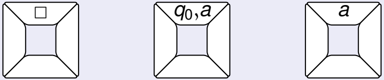
## Proof continued... (the starting row)

For input word $x = a_1 \cdots a_k$ we choose $n = 2p(k) + 1$.
(Assume $p(k) \geq k$, otherwise make it so.)

As first row we choose:




Tiles for building the first row (for every $a \in \Sigma$):

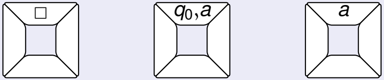



continued on the next slide...

# Bounded Tiling Problem is NP-complete

## Proof continued… (the types of tiles)

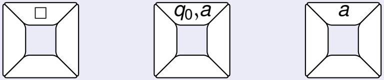
Tiles for building the first row (for every $a \in \Sigma$):



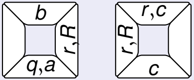
# Bounded Tiling Problem is NP-complete

## Proof continued... (the types of tiles)

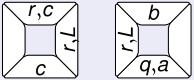
Tiles for building the first row (for every $a \in \Sigma$):




Tiles simulating the computation of $M$ (for every $c \in \Gamma$):




for $(r, b, R) \in \delta(q, a)$

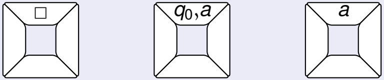



for $(r, b, L) \in \delta(q, a)$

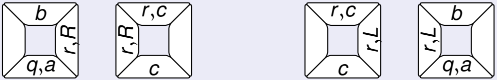
# Bounded Tiling Problem is NP-complete

## Proof continued... (the types of tiles)

Tiles for building the first row (for every $a \in \Sigma$):

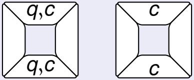



Tiles simulating the computation of $M$ (for every $c \in \Gamma$):




for $(r, b, R) \in \delta(q, a)$ for $(r, b, L) \in \delta(q, a)$

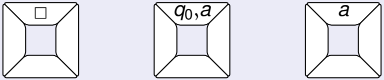
Tiles for leaving the tape unchanged (for every $q \in F$, $c \in \Gamma$):



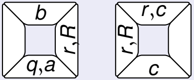
# Bounded Tiling Problem is NP-complete

## Proof continued… (the types of tiles)

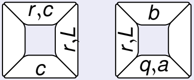
Tiles for building the first row (for every $a \in \Sigma$):




Tiles simulating the computation of $M$ (for every $c \in \Gamma$):

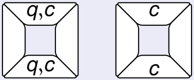



for $(r, b, R) \in \delta(q, a)$




for $(r, b, L) \in \delta(q, a)$

Tiles for leaving the tape unchanged (for every $q \in F$, $c \in \Gamma$):




continued on the next slide…

# Bounded Tiling Problem is NP-complete

## Proof continued. . .

Then, for input $x = a_1 \cdots a_k$ and with the indicated starting row:

$$n \times n \text{ field can be tiled} \quad \iff \quad x \in L(M)$$

# Bounded Tiling Problem is NP-complete

## Proof continued...

Then, for input $x = a_1 \cdots a_k$ and with the indicated starting row:

$$n \times n \text{ field can be tiled} \quad \iff \quad x \in L(M)$$

Every tiling simulates a computation of $M$ on input $x$.

# Bounded Tiling Problem is NP-complete

## Proof continued...

Then, for input $x = a_1 \cdots a_k$ and with the indicated starting row:

$$n \times n \text{ field can be tiled} \quad \Longleftrightarrow \quad x \in L(M)$$

Every tiling simulates a computation of $M$ on input $x$.

The computation takes at most $p(k)$ steps.

# Bounded Tiling Problem is NP-complete

## Proof continued...

Then, for input $x = a_1 \cdots a_k$ and with the indicated starting row:

$$n \times n \text{ field can be tiled} \quad \Longleftrightarrow \quad x \in L(M)$$

Every tiling simulates a computation of $M$ on input $x$.

The computation takes at most $p(k)$ steps.

So the computation fills only $p(k) < n$ rows of the tiling.

# Bounded Tiling Problem is NP-complete

## Proof continued...

Then, for input $x = a_1 \cdots a_k$ and with the indicated starting row:

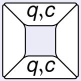
$$n \times n \text{ field can be tiled} \quad \iff \quad x \in L(M)$$

Every tiling simulates a computation of $M$ on input $x$.

The computation takes at most $p(k)$ steps.

So the computation fills only $p(k) < n$ rows of the tiling.

Hence, the $n \times n$ tiling can only be completed using




which exists only for $q \in F$.

# Bounded Tiling Problem is NP-complete

**Proof continued...**

Then, for input $x = a_1 \cdots a_k$ and with the indicated starting row:

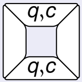
$$n \times n \text{ field can be tiled} \iff x \in L(M)$$

Every tiling simulates a computation of $M$ on input $x$.

The computation takes at most $p(k)$ steps.

So the computation fills only $p(k) < n$ rows of the tiling.

Hence, the $n \times n$ tiling can only be completed using




which exists only for $q \in F$.

Tiling can be finished

$$\iff M \text{ has an accepting computation for input } x.$$

# Example

Consider the TM $M$ with $\Sigma = \{a, b\}$, $\Gamma = \Sigma \cup \{\Box\}$, $F = \{q_1\}$ and

$$\delta(q_0, a) = \{(q_0, b, R)\} \qquad \delta(q_0, b) = \{(q_1, b, L)\}$$

Note that $L(M) = L(a^* b(a+b)^*) = L((a+b)^* b(a+b)^*)$

# Example

Consider the TM $M$ with $\Sigma = \{a, b\}$, $\Gamma = \Sigma \cup \{\Box\}$, $F = \{q_1\}$ and

$$\delta(q_0, a) = \{(q_0, b, R)\} \qquad \delta(q_0, b) = \{(q_1, b, L)\}$$

Note that $L(M) = L(a^* b(a+b)^*) = L((a+b)^* b(a+b)^*)$

For input $x$, $M$ takes at most $|x|$ steps. So we take $p(k) = k$.

# Example

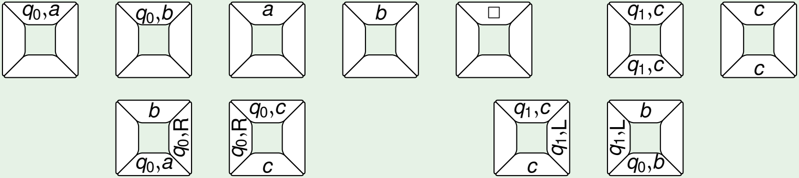
Consider the TM $M$ with $\Sigma = \{a, b\}$, $\Gamma = \Sigma \cup \{\Box\}$, $F = \{q_1\}$ and

$$\delta(q_0, a) = \{(q_0, b, R)\} \qquad \delta(q_0, b) = \{(q_1, b, L)\}$$

Note that $L(M) = L(a^* b(a + b)^*) = L((a + b)^* b(a + b)^*)$

For input $x$, $M$ takes at most $|x|$ steps. So we take $p(k) = k$.

The **tile types** are:




for every $c \in \Gamma$.

# Example

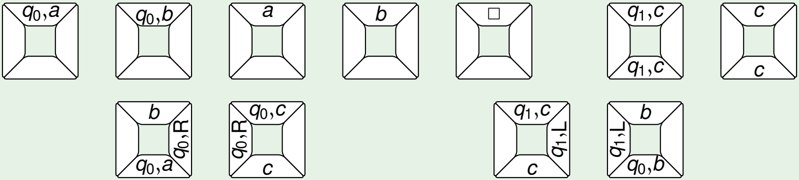
Consider the TM $M$ with $\Sigma = \{a, b\}$, $\Gamma = \Sigma \cup \{\Box\}$, $F = \{q_1\}$ and

$$\delta(q_0, a) = \{(q_0, b, R)\} \qquad \delta(q_0, b) = \{(q_1, b, L)\}$$

Note that $L(M) = L(a^* b(a+b)^*) = L((a+b)^* b(a+b)^*)$

For input $x$, $M$ takes at most $|x|$ steps. So we take $p(k) = k$.

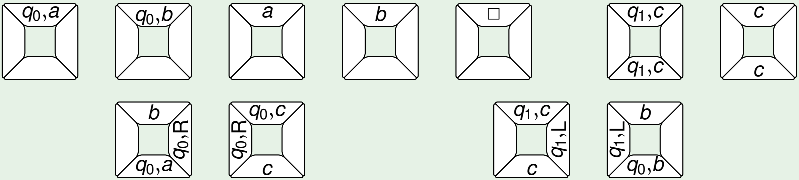
The **tile types** are:




for every $c \in \Gamma$.

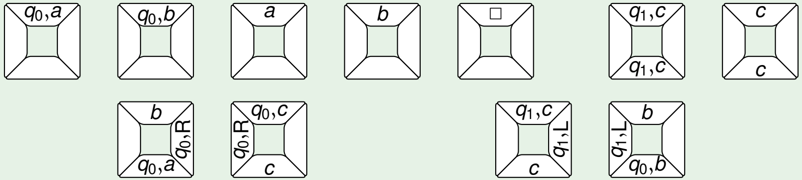
continued on the next slide...

# Example

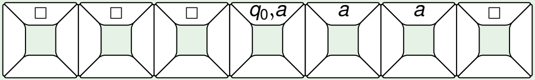



Consider the input word $aaa \notin L(M)$. Then $n = 2p(3) + 1 = 7$.

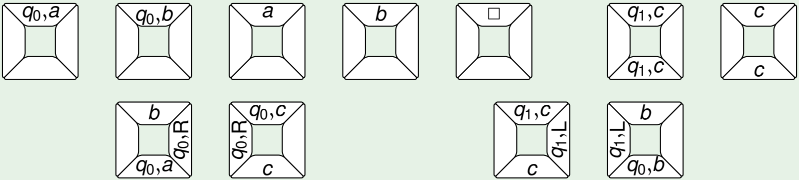
# Example



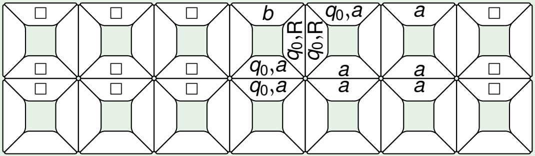

Consider the input word $aaa \notin L(M)$. Then $n = 2p(3) + 1 = 7$.



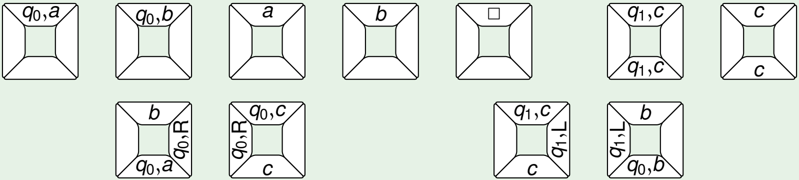
# Example



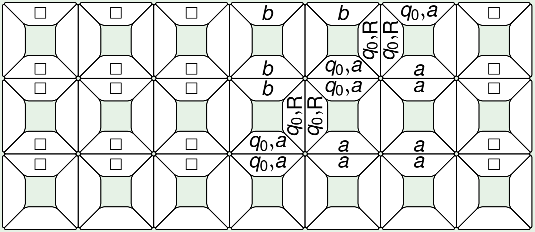

Consider the input word $\color{red}{aaa \notin L(M)}$. Then $n = 2p(3) + 1 = 7$.



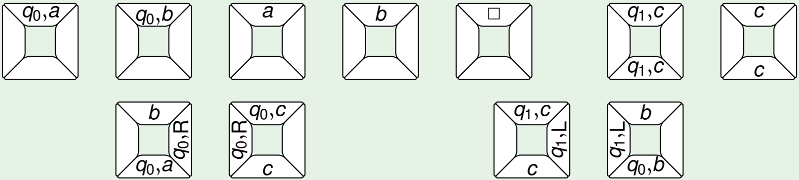
# Example



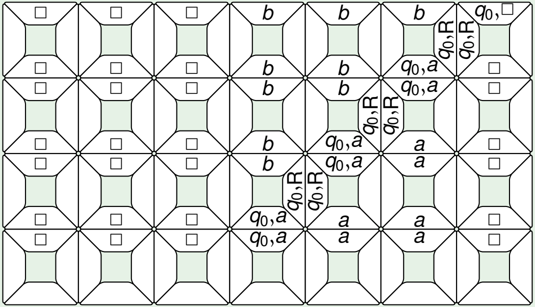

Consider the input word $aaa \notin L(M)$. Then $n = 2p(3) + 1 = 7$.



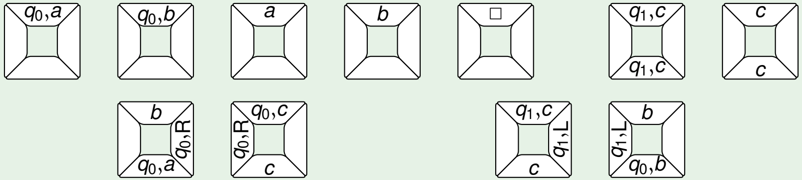
# Example



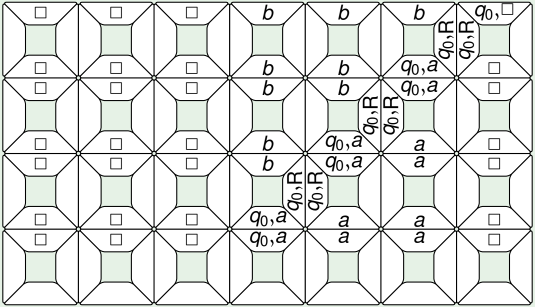

Consider the input word $\color{red}{aaa \notin L(M)}$. Then $n = 2p(3) + 1 = 7$.



# Example




Consider the input word $aaa \notin L(M)$. Then $n = 2p(3) + 1 = 7$.



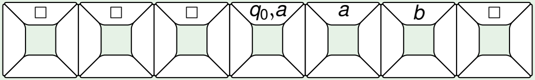

The tiling cannot be completed.

# Example continued

Consider the input word $aab \in L(M)$. Then $n = 2p(3) + 1 = 7$.

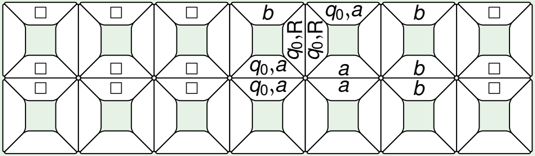
# Example continued

Consider the input word $aab \in L(M)$. Then $n = 2p(3) + 1 = 7$.



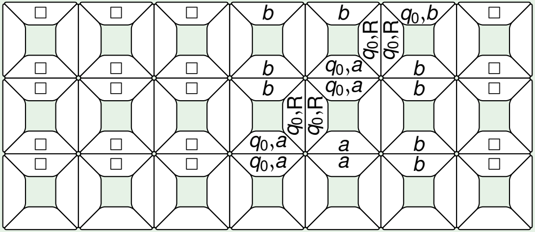
# Example continued

Consider the input word $aab \in L(M)$. Then $n = 2p(3) + 1 = 7$.



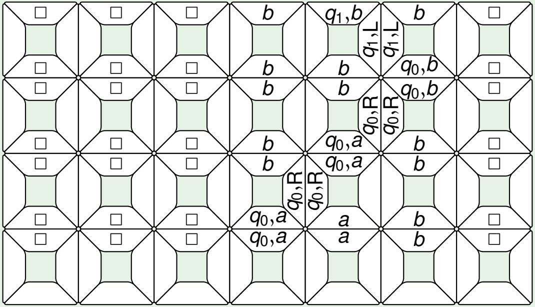
# Example continued

Consider the input word $aab \in L(M)$. Then $n = 2p(3) + 1 = 7$.



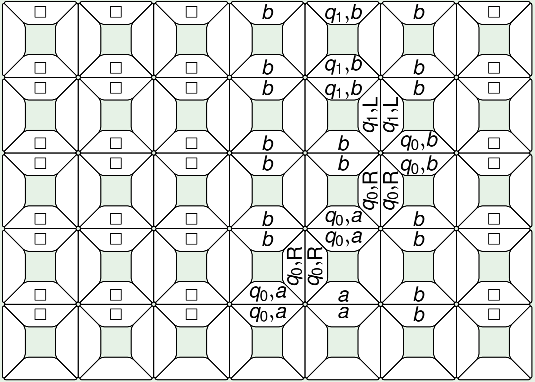
# Example continued

Consider the input word $aab \in L(M)$. Then $n = 2p(3) + 1 = 7$.



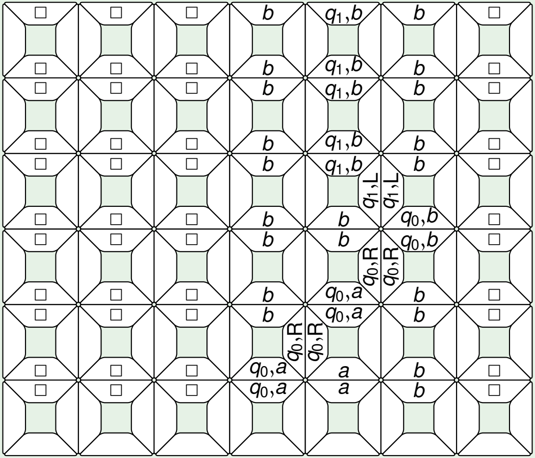
# Example continued

Consider the input word $aab \in L(M)$. Then $n = 2p(3) + 1 = 7$.



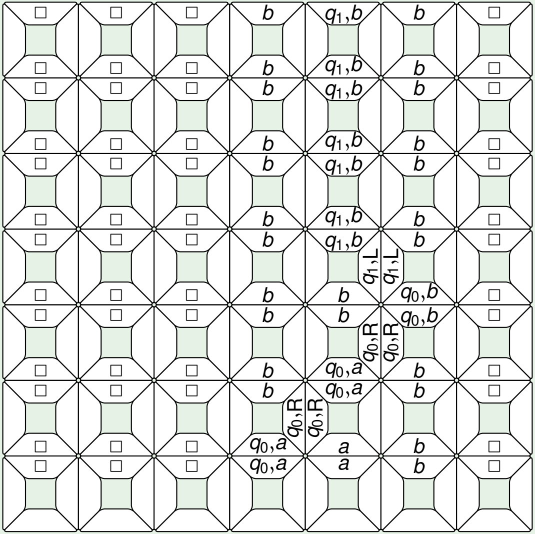
# Example continued

Consider the input word $aab \in L(M)$. Then $n = 2p(3) + 1 = 7$.



# Example continued

Consider the input word *aab* $\in L(M)$. Then $n = 2p(3) + 1 = 7$.




Complete tiling of the $7 \times 7$ field.

# Satisfiability Problem

# Satisfiability Problem is NP-complete

## Theorem of Cook

The satisfiability problem in propositional logic is NP-complete.

# Satisfiability Problem is NP-complete

## Theorem of Cook

The satisfiability problem in propositional logic is NP-complete.

## Proof

We give a polynomial-time reduction from the bounded tiling problem to the satisfiability problem.

# Satisfiability Problem is NP-complete

## Theorem of Cook

The satisfiability problem in propositional logic is NP-complete.

## Proof

We give a polynomial-time reduction from the bounded tiling problem to the satisfiability problem.

Given

- a set $T$ of tile types,
- a number $n$,
- a first row of tiles $t_1 \cdots t_n$.

# Satisfiability Problem is NP-complete

## Theorem of Cook

The satisfiability problem in propositional logic is NP-complete.

## Proof

We give a polynomial-time reduction from the bounded tiling problem to the satisfiability problem.

Given

- a set $T$ of tile types,
- a number $n$,
- a first row of tiles $t_1 \cdots t_n$.

We create a satisfiability problem as follows.

# Satisfiability Problem is NP-complete

## Theorem of Cook

The satisfiability problem in propositional logic is NP-complete.

## Proof

We give a polynomial-time reduction from the bounded tiling problem to the satisfiability problem.

Given

- a set $T$ of tile types,
- a number $n$,
- a first row of tiles $t_1 \cdots t_n$.

We create a satisfiability problem as follows.

We introduce Boolean variables $x_{rct}$ for $1 \leq r, c \leq n$ and $t \in T$.

Intention: $x_{rct} = \texttt{true} \iff$ tile of type $t$ at row $r$ and column $c$.

# Satisfiability Problem is NP-complete

## Theorem of Cook

The satisfiability problem in propositional logic is NP-complete.

## Proof

We give a polynomial-time reduction from the bounded tiling problem to the satisfiability problem.

Given

- a set $T$ of tile types,
- a number $n$,
- a first row of tiles $t_1 \cdots t_n$.

We create a satisfiability problem as follows.

We introduce Boolean variables $x_{rct}$ for $1 \leq r, c \leq n$ and $t \in T$.

Intention: $x_{rct} = \texttt{true} \iff$ tile of type $t$ at row $r$ and column $c$.

continued on the next slide...

# Satisfiability Problem is NP-complete

## Proof continued…

We define $\Phi$ to be the conjunction of the 4 formulas:

1. Fist row is $t_1 \cdots t_n$: $\bigwedge_{c=1}^{n} x_{1ct_k}$
2. At every position at most one tile type:
$$\bigwedge_{r=1}^{n} \bigwedge_{c=1}^{n} \bigwedge_{t \neq t'} \neg(x_{rct} \wedge x_{rct'})$$
3. Neighbouring tiles must match (horizontal neighbours):
$$\bigwedge_{r=1}^{n} \bigwedge_{c=1}^{n-1} \bigvee_{tt' \text{ matches}} (x_{rct} \wedge x_{r(c+1)t'})$$
4. Neighbouring tiles must match (vertical neighbours):
$$\bigwedge_{r=1}^{n-1} \bigwedge_{c=1}^{n} \bigvee_{\substack{t' \\ t} \text{ matches}} (x_{rct} \wedge x_{(r+1)ct'})$$

# Satisfiability Problem is NP-complete

## Proof continued...

We define $\Phi$ to be the conjunction of the 4 formulas:

1. Fist row is $t_1 \cdots t_n$: $\bigwedge_{c=1}^{n} x_{1ct_k}$
2. At every position at most one tile type:
$$\bigwedge_{r=1}^{n} \bigwedge_{c=1}^{n} \bigwedge_{t \neq t'} \neg(x_{rct} \wedge x_{rct'})$$
3. Neighbouring tiles must match (horizontal neighbours):
$$\bigwedge_{r=1}^{n} \bigwedge_{c=1}^{n-1} \bigvee_{tt' \text{ matches}} (x_{rct} \wedge x_{r(c+1)t'})$$
4. Neighbouring tiles must match (vertical neighbours):
$$\bigwedge_{r=1}^{n-1} \bigwedge_{c=1}^{n} \bigvee_{\substack{t' \\ t} \text{ matches}} (x_{rct} \wedge x_{(r+1)ct'})$$

Size of the formula is polynomial in $n$.

# Satisfiability Problem is NP-complete

## Proof continued…

We define $\Phi$ to be the conjunction of the 4 formulas:

1. Fist row is $t_1 \cdots t_n$: $\bigwedge_{c=1}^{n} x_{1ct_k}$
2. At every position at most one tile type:
$$\bigwedge_{r=1}^{n} \ \bigwedge_{c=1}^{n} \ \bigwedge_{t \neq t'} \neg(x_{rct} \wedge x_{rct'})$$
3. Neighbouring tiles must match (horizontal neighbours):
$$\bigwedge_{r=1}^{n} \ \bigwedge_{c=1}^{n-1} \ \bigvee_{tt' \text{ matches}} (x_{rct} \wedge x_{r(c+1)t'})$$
4. Neighbouring tiles must match (vertical neighbours):
$$\bigwedge_{r=1}^{n-1} \ \bigwedge_{c=1}^{n} \ \bigvee_{\substack{t' \\ t} \text{ matches}} (x_{rct} \wedge x_{(r+1)ct'})$$

Size of the formula is polynomial in $n$.

There exists an $n \times n$ tiling with first row $t_1 \cdots t_n$
$\iff$ the propositional formula $\Phi$ is satisfiable.

# Satisfiability Problem is NP-complete

## Proof continued...

We define $\Phi$ to be the conjunction of the 4 formulas:

1. Fist row is $t_1 \cdots t_n$: $\quad \bigwedge_{c=1}^{n} x_{1ct_k}$
2. At every position at most one tile type:
$$\bigwedge_{r=1}^{n} \; \bigwedge_{c=1}^{n} \; \bigwedge_{t \neq t'} \neg(x_{rct} \wedge x_{rct'})$$
3. Neighbouring tiles must match (horizontal neighbours):
$$\bigwedge_{r=1}^{n} \; \bigwedge_{c=1}^{n-1} \; \bigvee_{tt' \text{ matches}} (x_{rct} \wedge x_{r(c+1)t'})$$
4. Neighbouring tiles must match (vertical neighbours):
$$\bigwedge_{r=1}^{n-1} \; \bigwedge_{c=1}^{n} \; \bigvee_{\substack{t' \\ t} \text{ matches}} (x_{rct} \wedge x_{(r+1)ct'})$$

Size of the formula is polynomial in $n$.

There exists an $n \times n$ tiling with first row $t_1 \cdots t_n$

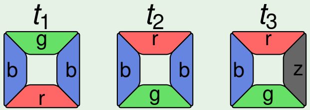
$\iff$ the propositional formula $\Phi$ is satisfiable.

Thus we have a polynomial-time reduction. $\square$

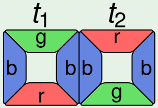
# Exercise

Reduce this bounded tiling problem to the satisfiability problem.

Types of tiles:



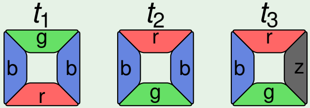

First row:



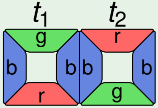
# Exercise

Reduce this bounded tiling problem to the satisfiability problem.

Types of tiles:




First row:



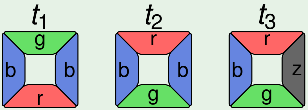

Then $\Phi$ is the conjunction of:

1. $x_{11t_1} \wedge x_{12t_2}$

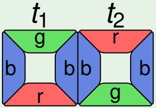
# Exercise

Reduce this bounded tiling problem to the satisfiability problem.

Types of tiles:




First row:



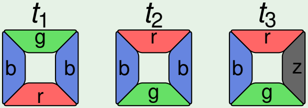

Then $\Phi$ is the conjunction of:

1. $x_{11t_1} \wedge x_{12t_2}$
2. $\neg(x_{11t_1} \wedge x_{11t_2}) \wedge \neg(x_{12t_1} \wedge x_{12t_2}) \wedge \neg(x_{21t_1} \wedge x_{21t_2}) \wedge \neg(x_{22t_1} \wedge x_{22t_2}) \wedge$
   $\neg(x_{11t_1} \wedge x_{11t_3}) \wedge \neg(x_{12t_1} \wedge x_{12t_3}) \wedge \neg(x_{21t_1} \wedge x_{21t_3}) \wedge \neg(x_{22t_1} \wedge x_{22t_3}) \wedge$
   $\neg(x_{11t_2} \wedge x_{11t_3}) \wedge \neg(x_{12t_2} \wedge x_{12t_3}) \wedge \neg(x_{21t_2} \wedge x_{21t_3}) \wedge \neg(x_{22t_2} \wedge x_{22t_3})$

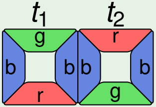
# Exercise

Reduce this bounded tiling problem to the satisfiability problem.

Types of tiles:




First row:



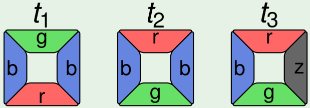

Then $\Phi$ is the conjunction of:

1. $x_{11t_1} \wedge x_{12t_2}$
2. $\neg(x_{11t_1} \wedge x_{11t_2}) \wedge \neg(x_{12t_1} \wedge x_{12t_2}) \wedge \neg(x_{21t_1} \wedge x_{21t_2}) \wedge \neg(x_{22t_1} \wedge x_{22t_2}) \wedge$
   $\neg(x_{11t_1} \wedge x_{11t_3}) \wedge \neg(x_{12t_1} \wedge x_{12t_3}) \wedge \neg(x_{21t_1} \wedge x_{21t_3}) \wedge \neg(x_{22t_1} \wedge x_{22t_3}) \wedge$
   $\neg(x_{11t_2} \wedge x_{11t_3}) \wedge \neg(x_{12t_2} \wedge x_{12t_3}) \wedge \neg(x_{21t_2} \wedge x_{21t_3}) \wedge \neg(x_{22t_2} \wedge x_{22t_3})$
3. $\big((x_{11t_1} \wedge x_{12t_1}) \vee (x_{11t_1} \wedge x_{12t_2}) \vee (x_{11t_1} \wedge x_{12t_3}) \vee$
   $(x_{11t_2} \wedge x_{12t_1}) \vee (x_{11t_2} \wedge x_{12t_2}) \vee (x_{11t_2} \wedge x_{12t_3})\big) \wedge$
   $\big((x_{21t_1} \wedge x_{22t_1}) \vee (x_{21t_1} \wedge x_{22t_2}) \vee (x_{21t_1} \wedge x_{22t_3}) \vee$
   $(x_{21t_2} \wedge x_{22t_1}) \vee (x_{21t_2} \wedge x_{22t_2}) \vee (x_{21t_2} \wedge x_{22t_3})\big)$

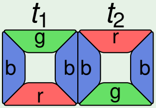
# Exercise

Reduce this bounded tiling problem to the satisfiability problem.

Types of tiles:




First row:




Then $\Phi$ is the conjunction of:

1. $x_{11t_1} \wedge x_{12t_2}$
2. $\neg(x_{11t_1} \wedge x_{11t_2}) \wedge \neg(x_{12t_1} \wedge x_{12t_2}) \wedge \neg(x_{21t_1} \wedge x_{21t_2}) \wedge \neg(x_{22t_1} \wedge x_{22t_2}) \wedge$
   $\neg(x_{11t_1} \wedge x_{11t_3}) \wedge \neg(x_{12t_1} \wedge x_{12t_3}) \wedge \neg(x_{21t_1} \wedge x_{21t_3}) \wedge \neg(x_{22t_1} \wedge x_{22t_3}) \wedge$
   $\neg(x_{11t_2} \wedge x_{11t_3}) \wedge \neg(x_{12t_2} \wedge x_{12t_3}) \wedge \neg(x_{21t_2} \wedge x_{21t_3}) \wedge \neg(x_{22t_2} \wedge x_{22t_3})$
3. $\big((x_{11t_1} \wedge x_{12t_1}) \vee (x_{11t_1} \wedge x_{12t_2}) \vee (x_{11t_1} \wedge x_{12t_3}) \vee$
   $(x_{11t_2} \wedge x_{12t_1}) \vee (x_{11t_2} \wedge x_{12t_2}) \vee (x_{11t_2} \wedge x_{12t_3})\big) \wedge$
   $\big((x_{21t_1} \wedge x_{22t_1}) \vee (x_{21t_1} \wedge x_{22t_2}) \vee (x_{21t_1} \wedge x_{22t_3}) \vee$
   $(x_{21t_2} \wedge x_{22t_1}) \vee (x_{21t_2} \wedge x_{22t_2}) \vee (x_{21t_2} \wedge x_{22t_3})\big)$
4. $\big((x_{11t_1} \wedge x_{21t_2}) \vee (x_{11t_1} \wedge x_{21t_3}) \vee (x_{11t_2} \wedge x_{21t_1}) \vee (x_{11t_3} \wedge x_{21t_1})\big) \wedge$
   $\big((x_{12t_1} \wedge x_{22t_2}) \vee (x_{12t_1} \wedge x_{22t_3}) \vee (x_{12t_2} \wedge x_{22t_1}) \vee (x_{12t_3} \wedge x_{22t_1})\big)$

P = NP?

# NP-completeness and P = NP?

**Theorem**

If an NP-complete language $L$ is also in $P$, then P $=$ NP.

# NP-completeness and P = NP?

**Theorem**

If an NP-complete language $L$ is also in $P$, then P $=$ NP.

**Proof.**

Assume that $L$ is NP-complete and in P.

# NP-completeness and P = NP?

## Theorem

If an NP-complete language $L$ is also in $P$, then P $=$ NP.

## Proof.

Assume that $L$ is NP-complete and in P.

Let $L' \in$ NP.

# NP-completeness and P = NP?

## Theorem

If an NP-complete language $L$ is also in $P$, then $\mathrm{P} = \mathrm{NP}$.

## Proof.

Assume that $L$ is NP-complete and in P.

Let $L' \in \mathrm{NP}$.

As $L$ is NP-complete, there is a polynomial-time reduction $f$ with

$$x \in L' \iff f(x) \in L$$

# NP-completeness and P = NP?

**Theorem**

If an NP-complete language $L$ is also in $P$, then $\mathrm{P} = \mathrm{NP}$.

**Proof.**

Assume that $L$ is NP-complete and in P.

Let $L' \in \mathrm{NP}$.

As $L$ is NP-complete, there is a polynomial-time reduction $f$ with

$$x \in L' \iff f(x) \in L$$

Since $L \in \mathrm{P}$, we can compute $f(x) \in L$ in polynomial time.

# NP-completeness and P = NP?

**Theorem**

If an NP-complete language $L$ is also in $P$, then $\mathrm{P} = \mathrm{NP}$.

**Proof.**

Assume that $L$ is NP-complete and in P.

Let $L' \in \mathrm{NP}$.

As $L$ is NP-complete, there is a polynomial-time reduction $f$ with

$$x \in L' \iff f(x) \in L$$

Since $L \in \mathrm{P}$, we can compute $f(x) \in L$ in polynomial time.

Thus $x \in L'$ can be decided in polynomial time.

Hence $L' \in \mathrm{P}$. $\square$

# NP-completeness and P = NP?

**Theorem**

If an NP-complete language $L$ is also in $P$, then $\mathrm{P} = \mathrm{NP}$.

**Proof.**

Assume that $L$ is NP-complete and in P.

Let $L' \in \mathrm{NP}$.

As $L$ is NP-complete, there is a polynomial-time reduction $f$ with

$$x \in L' \iff f(x) \in L$$

Since $L \in \mathrm{P}$, we can compute $f(x) \in L$ in polynomial time.

Thus $x \in L'$ can be decided in polynomial time.

Hence $L' \in \mathrm{P}$. □

For proving $\mathrm{P} = \mathrm{NP}$ it suffices to show that one NP-complete problem can be solved in deterministic polynomial time.

# co-NP

# The Class co-NP

A problem $L$ is in co-NP if the complement $\overline{L}$ is in NP.

In other words, the set of instances without solution is in NP.

# The Class co-NP

A problem $L$ is in co-NP if the complement $\overline{L}$ is in NP.

In other words, the set of instances without solution is in NP.

The question whether a propositional formula is **not** satisfiable is in co-NP.

# The Class co-NP

A problem $L$ is in co-NP if the complement $\overline{L}$ is in NP.

In other words, the set of instances without solution is in NP.

The question whether a propositional formula is **not** satisfiable is in co-NP.

It is unknown whether NP $=$ co-NP.

# The Class co-NP

A problem $L$ is in co-NP if the complement $\overline{L}$ is in NP.

In other words, the set of instances without solution is in NP.

The question whether a propositional formula is **not** satisfiable is in co-NP.

It is unknown whether NP $=$ co-NP.

It is unknown whether the satisfiability problem is in co-NP.

# The Class co-NP

A problem $L$ is in co-NP if the complement $\overline{L}$ is in NP.

In other words, the set of instances without solution is in NP.

The question whether a propositional formula is **not** satisfiable is in co-NP.

It is unknown whether NP $=$ co-NP.

It is unknown whether the satisfiability problem is in co-NP.

The difficulty is that it has to be shown that a formula evaluates to `false` for **every** variable assignment.

# The Class co-NP

A problem $L$ is in co-NP if the complement $\overline{L}$ is in NP.

In other words, the set of instances without solution is in NP.

The question whether a propositional formula is **not** satisfiable is in co-NP.

It is unknown whether NP $=$ co-NP.

It is unknown whether the satisfiability problem is in co-NP.

The difficulty is that it has to be shown that a formula evaluates to `false` for **every** variable assignment.

**Theorem**

If an NP-complete problem is in co-NP, then NP $=$ co-NP.

Note that there are problems that are both in NP $\cap$ co-NP.

# Space Complexity

# Space Complexity

Let $f, g : \mathbb{N} \to \mathbb{N}$.

A nondeterministic Turing machine $M$

**runs in space $f$**

if for every input $w$, every computation of $M$ visits at most $f(|w|)$ positions on the tape.

# Space Complexity

Let $f, g : \mathbb{N} \to \mathbb{N}$.

A nondeterministic Turing machine $M$

**runs in space $f$**

if for every input $w$, every computation of $M$ visits at most $f(|w|)$ positions on the tape.

The function $f$ gives an upper bound on the number of visited cells on the tape in terms of the length of the input word.

# Complexity Classes PSpace and NPSpace

A nondeterministic Turing machine $M$ is **polynomial space** if $M$ runs in space $p$ for some polynomial $p$.

# Complexity Classes PSpace and NPSpace

A nondeterministic Turing machine $M$ is **polynomial space** if $M$ runs in space $p$ for some polynomial $p$.

$$\textbf{NPSpace} = \{\, L(M) \mid M \text{ nondeterministic polynomial space TM} \,\}$$
$$\textbf{PSpace} = \{\, L(M) \mid M \text{ deterministic polynomial space TM} \,\}$$

# Complexity Classes PSpace and NPSpace

A nondeterministic Turing machine $M$ is **polynomial space** if $M$ runs in space $p$ for some polynomial $p$.

$$\textbf{NPSpace} = \{\, L(M) \mid M \text{ nondeterministic polynomial space TM} \,\}$$
$$\textbf{PSpace} = \{\, L(M) \mid M \text{ deterministic polynomial space TM} \,\}$$

$$\text{P} \subseteq \text{PSpace} \qquad\qquad \text{NP} \subseteq \text{NPSpace}$$

# Complexity Classes PSpace and NPSpace

A nondeterministic Turing machine $M$ is **polynomial space** if $M$ runs in space $p$ for some polynomial $p$.

$$\textbf{NPSpace} = \{\, L(M) \mid M \text{ nondeterministic polynomial space TM} \,\}$$
$$\textbf{PSpace} = \{\, L(M) \mid M \text{ deterministic polynomial space TM} \,\}$$

$$\mathrm{P} \subseteq \mathrm{PSpace} \qquad\qquad \mathrm{NP} \subseteq \mathrm{NPSpace}$$

**Theorem of Savitch**

$$\mathrm{PSpace} = \mathrm{NPSpace}$$

# Complexity Classes PSpace and NPSpace

A nondeterministic Turing machine $M$ is **polynomial space** if $M$ runs in space $p$ for some polynomial $p$.

$$\textbf{NPSpace} = \{\, L(M) \mid M \text{ nondeterministic polynomial space TM} \,\}$$
$$\textbf{PSpace} = \{\, L(M) \mid M \text{ deterministic polynomial space TM} \,\}$$

$$\text{P} \subseteq \text{PSpace} \qquad\qquad \text{NP} \subseteq \text{NPSpace}$$

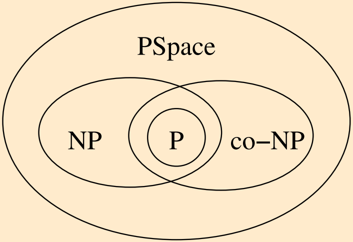
## Theorem of Savitch

$$\text{PSpace} = \text{NPSpace}$$

Actually, the theorem says something more general:

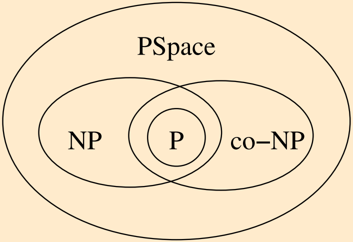
If $L$ is accepted by a nondeterministic TM in $f(n)$ space,
then $L$ is accepted by a deterministic TM in $f(n)^2$ space.

# PSpace-completeness




It is unknown whether these inclusions are strict.

# PSpace-completeness




It is unknown whether these inclusions are strict.

A language $L \in$ PSpace is **PSpace-complete** if every language $L' \in$ PSpace is polynomial-time reducible to $L$.

$L(r) = \Sigma^*$ ? for regular expression $r$ is PSpace-complete.

# The Classes EXP, NEXP and EXPSpace

# The Classes EXP and NEXP

A nondeterministic Turing machine $M$ is

- **exponential time** if $M$ runs in time $2^{p(|x|)}$ and
- **exponential space** if $M$ runs in space $2^{p(|x|)}$

for some polynomial $p$.

# The Classes EXP and NEXP

A nondeterministic Turing machine $M$ is

- **exponential time** if $M$ runs in time $2^{p(|x|)}$ and
- **exponential space** if $M$ runs in space $2^{p(|x|)}$

for some polynomial $p$.

$$\begin{aligned}
\textbf{NEXP} &= \{\, L(M) \mid M \text{ nondeterm. exponential time TM} \,\} \\
\textbf{EXP} &= \{\, L(M) \mid M \text{ deterministic exponential time TM} \,\} \\
\textbf{NEXPSpace} &= \{\, L(M) \mid M \text{ nondeterm. exponential space TM} \,\} \\
\textbf{EXPSpace} &= \{\, L(M) \mid M \text{ deterministic exponential space TM} \,\}
\end{aligned}$$

# The Classes EXP and NEXP

A nondeterministic Turing machine $M$ is

- **exponential time** if $M$ runs in time $2^{p(|x|)}$ and
- **exponential space** if $M$ runs in space $2^{p(|x|)}$

for some polynomial $p$.

$$\begin{aligned}
\textbf{NEXP} &= \{\, L(M) \mid M \text{ nondeterm. exponential time TM} \,\} \\
\textbf{EXP} &= \{\, L(M) \mid M \text{ deterministic exponential time TM} \,\} \\
\textbf{NEXPSpace} &= \{\, L(M) \mid M \text{ nondeterm. exponential space TM} \,\} \\
\textbf{EXPSpace} &= \{\, L(M) \mid M \text{ deterministic exponential space TM} \,\}
\end{aligned}$$

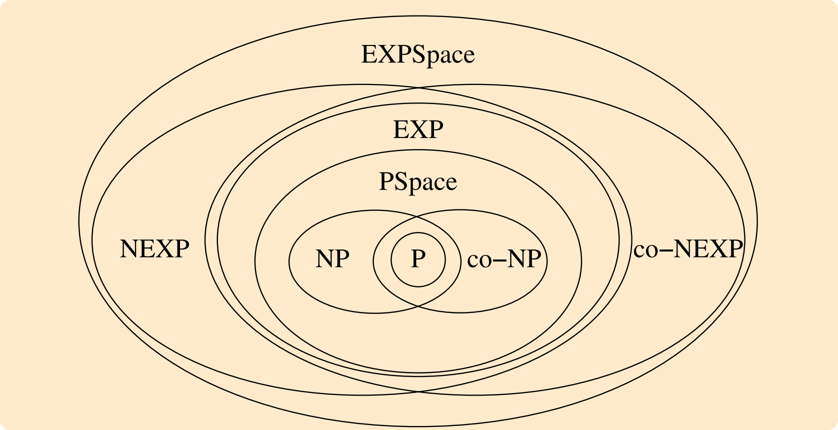
$$\text{P} \subseteq \text{NP} \subseteq \text{PSpace} \subseteq \text{EXP} \subseteq \text{NEXP} \subseteq \text{EXPSpace}$$

It is unknown whether these inclusions are strict. We know

$$\text{P} \neq \text{EXP} \quad \text{NP} \neq \text{NEXP} \quad \text{PSpace} \neq \text{EXPSpace} = \text{NEXPSpace}$$

PSpace $\subseteq$ EXP holds since a polynomial-space TM can at most take an exponential number of configurations.

# Complexity Hierarchy




The following inclusions are known to be strict:

$P \neq EXP \qquad NP \neq NEXP \qquad PSpace \neq EXPSpace$