Complexity classes

IIC3242

Let us repeat the basics

We have defined various complexity classes. Now we want to see how they relate one to another.

Computation model: Turing machines (deterministic or nondeterministic) with various number of tapes.

- We defined the executions time of a machine.
- We defined the space complexity (here we used a machine with an input tape).

A technical assumption

To avoid unintuitive behaviour we will need the following assumption on the functions used to measure the complexity:

Assumption

A function f is a **proper complexity function** if f is non-decreasing and there exists a deterministic TM M (with one input tape and $k \geq 1$ work tapes) such that:

- The time complexity t_M of M is $O(n + f(n))$
- The space complexity s_M of M is $O(f(n))$
- M halts on all inputs
- When started on any input of length n the machine M will halt with the string $1^{f(n)}$ written on its final work tape.

From now on all the functions we use are assumed to be proper.

Deterministic classes: time complexity

Input alphabet: Σ.

DTIME(t): the set of all languages $L \subseteq \Sigma^*$ that are decided by some $O(t)$ time deterministic Turing machine.

Two fundamental classes:

$$\begin{aligned} \text{PTIME} &= \bigcup_{k \in \mathbb{N}} \text{DTIME}(n^k) \\ \text{EXPTIME} &= \bigcup_{k \in \mathbb{N}} \text{DTIME}(2^{n^k}) \end{aligned}$$

Deterministic classes: time complexity

Input alphabet: Σ.

DTIME(t): the set of all languages $L \subseteq \Sigma^*$ that are decided by some $O(t)$ time deterministic Turing machine.

Two fundamental classes:

$$\begin{aligned} \text{PTIME} &= \bigcup_{k \in \mathbb{N}} \text{DTIME}(n^k) \\ \text{EXPTIME} &= \bigcup_{k \in \mathbb{N}} \text{DTIME}(2^{n^k}) \end{aligned}$$

PTIME: The set of all *efficiently* solvable problems.

Nondeterministic classes: time complexity

NTIME(t): the set of all languages $L \subseteq \Sigma^*$ that are decided by some $O(t)$ time nondeterministic Turing machine.

Two fundamental classes:

$$
\begin{aligned}
\text{NP} &= \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k) \\
\text{NEXPTIME} &= \bigcup_{k \in \mathbb{N}} \text{NTIME}(2^{n^k})
\end{aligned}
$$

Deterministic classes: space complexity

DSPACE(s): the set of all languages $L \subseteq \Sigma^*$ that are decided by some $O(s)$ space deterministic Turing machine.

Three important classes:

$$
\begin{aligned}
\text{LOGSPACE} &= \text{DSPACE}(\log n) \\
\text{PSPACE} &= \bigcup_{k \in \mathbb{N}} \text{DSPACE}(n^k) \\
\text{EXPSPACE} &= \bigcup_{k \in \mathbb{N}} \text{DSPACE}(2^{n^k})
\end{aligned}
$$

Nondeterministic classes: time complexity

NSPACE(s): the set of all languages $L \subseteq \Sigma^*$ that are decided by some $O(s)$ space nondeterministic Turing machine.

Three important classes:

$$\begin{aligned} \text{NLOGSPACE} &= \text{NSPACE}(\log n) \\ \text{NPSPACE} &= \bigcup_{k \in \mathbb{N}} \text{NSPACE}(n^k) \\ \text{NEXPSPACE} &= \bigcup_{k \in \mathbb{N}} \text{NSPACE}(2^{n^k}) \end{aligned}$$

Complement of a complexity class

Given a language L over the alphabet Σ:

$$\overline{L} \;=\; \{w \in \Sigma^* \mid w \notin L\}.$$

Definition

Given a complexity class $\mathcal{C}$, the set of complements of languages in $\mathcal{C}$ is defined as: $\text{co-}\mathcal{C} \;=\; \{\overline{L} \mid L \in \mathcal{C}\}$

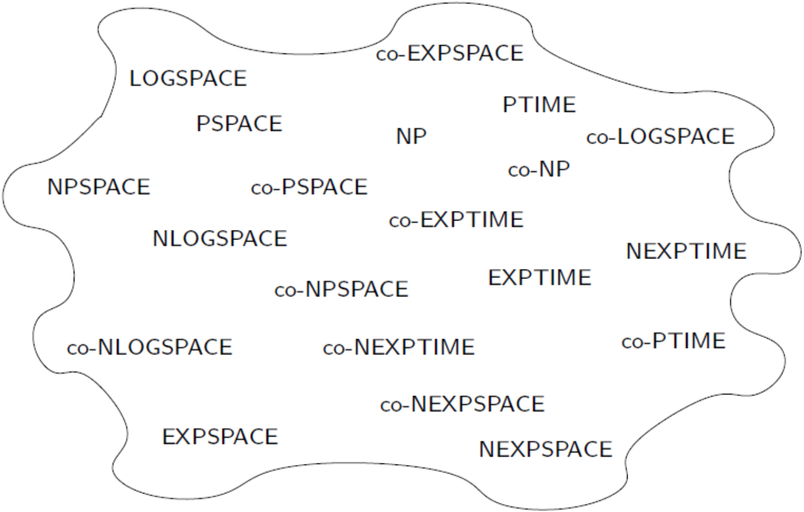
Example

A very well-known class: co-NP

- Give an example of a problem from this class.
- Is this class equal to NP?

Relations between complexity classes

We start with this:



Basic facts

The following results are straightforward:

Basic facts

The following results are straightforward:

- If $\mathcal{C}$ is a deterministic complexity class then co-$\mathcal{C} = \mathcal{C}$.

Basic facts

The following results are straightforward:

- If $\mathcal{C}$ is a deterministic complexity class then co-$\mathcal{C} = \mathcal{C}$.
- If f is $O(g)$ then:
  - $\text{DTIME}(f) \subseteq \text{DTIME}(g)$
  - $\text{NTIME}(f) \subseteq \text{NTIME}(g)$
  - $\text{DSPACE}(f) \subseteq \text{DSPACE}(g)$
  - $\text{NSPACE}(f) \subseteq \text{NSPACE}(g)$

Basic facts

The following results are straightforward:

- If $\mathcal{C}$ is a deterministic complexity class then co-$\mathcal{C} = \mathcal{C}$.
- If f is $O(g)$ then:
  - $\text{DTIME}(f) \subseteq \text{DTIME}(g)$
  - $\text{NTIME}(f) \subseteq \text{NTIME}(g)$
  - $\text{DSPACE}(f) \subseteq \text{DSPACE}(g)$
  - $\text{NSPACE}(f) \subseteq \text{NSPACE}(g)$
- $\text{DTIME}(f) \subseteq \text{NTIME}(f)$

Basic facts

The following results are straightforward:

- If $\mathcal{C}$ is a deterministic complexity class then co-$\mathcal{C} = \mathcal{C}$.
- If f is $O(g)$ then:
  - $\text{DTIME}(f) \subseteq \text{DTIME}(g)$
  - $\text{NTIME}(f) \subseteq \text{NTIME}(g)$
  - $\text{DSPACE}(f) \subseteq \text{DSPACE}(g)$
  - $\text{NSPACE}(f) \subseteq \text{NSPACE}(g)$
- $\text{DTIME}(f) \subseteq \text{NTIME}(f)$
- $\text{DSPACE}(f) \subseteq \text{NSPACE}(f)$

Basic facts

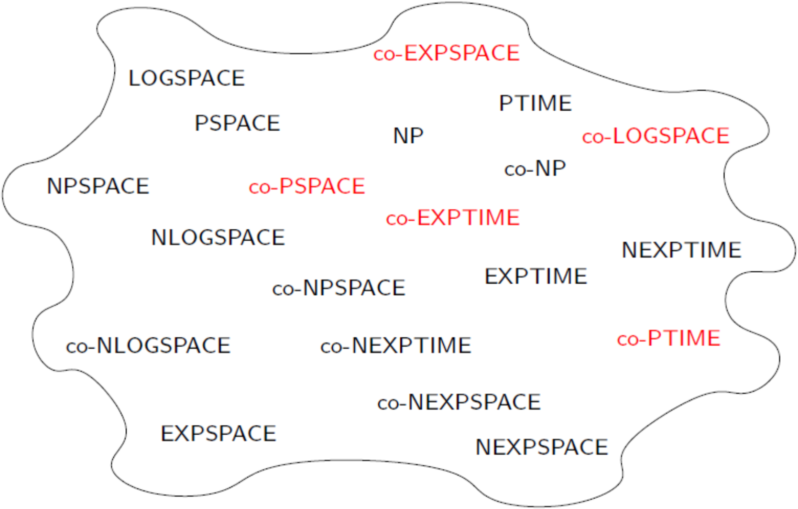
The following results are straightforward:

- If $\mathcal{C}$ is a deterministic complexity class then co-$\mathcal{C} = \mathcal{C}$.
- If f is $O(g)$ then:
  - $\text{DTIME}(f) \subseteq \text{DTIME}(g)$
  - $\text{NTIME}(f) \subseteq \text{NTIME}(g)$
  - $\text{DSPACE}(f) \subseteq \text{DSPACE}(g)$
  - $\text{NSPACE}(f) \subseteq \text{NSPACE}(g)$
- $\text{DTIME}(f) \subseteq \text{NTIME}(f)$
- $\text{DSPACE}(f) \subseteq \text{NSPACE}(f)$

So we can start reducing our figure

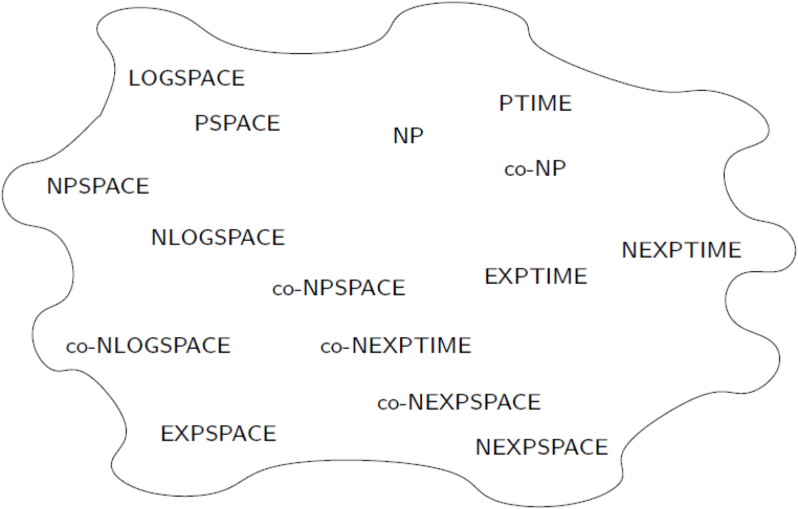
Relations between complexity classes

We can eliminate some classes:

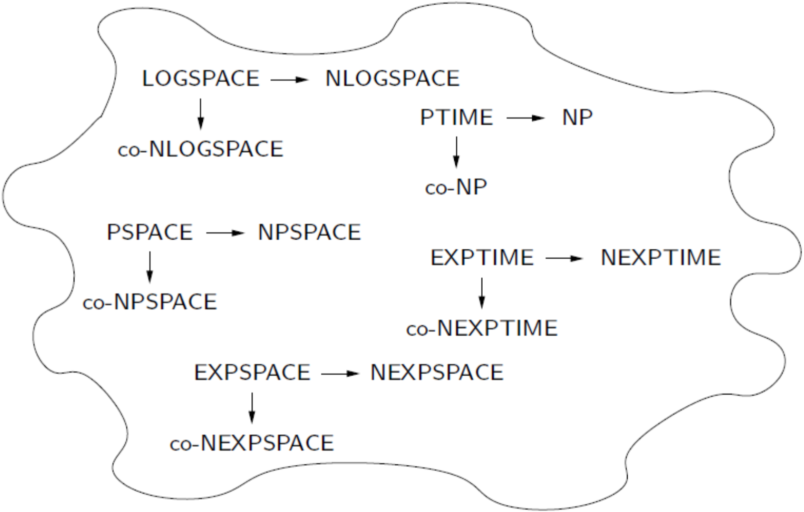


Relations between complexity classes

We can eliminate some classes:



We also have some inclusions:



Relations between time and space

Theorem

$$NTIME(f(n)) \subseteq DSPACE(f(n))$$

Relations between time and space

Theorem

$$NTIME(f(n)) \subseteq DSPACE(f(n))$$

Exercise

Prove this (we already said how).

Relations between time and space

Theorem

$$NTIME(f(n)) \quad \subseteq \quad DSPACE(f(n))$$

Exercise

Prove this (we already said how).

Corollary

$$\begin{aligned} NP \quad & \subseteq \quad PSPACE \\ co\text{-}NP \quad & \subseteq \quad PSPACE \\ NEXPTIME \quad & \subseteq \quad EXPSPACE \\ co\text{-}NEXPTIME \quad & \subseteq \quad EXPSPACE \end{aligned}$$

Relations between time and space

Theorem

If $f(n)$ is $\Omega(\log n)$ then:

$$NSPACE(f(n)) \subseteq \bigcup_{k \in \mathbb{N}} DTIME(2^{k \cdot f(n)})$$

Relations between time and space

Theorem

If $f(n)$ is $\Omega(\log n)$ then:

$$NSPACE(f(n)) \quad \subseteq \quad \bigcup_{k \in \mathbb{N}} DTIME(2^{k \cdot f(n)})$$

Exercise

Prove this.

- Why do we require that $f(n)$ is $\Omega(\log n)$?

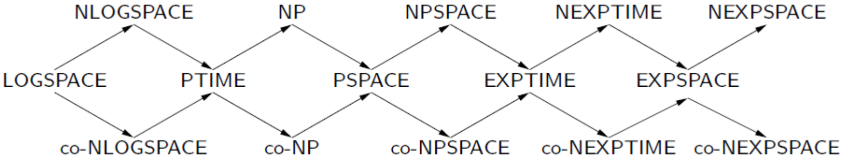
Relations between time and space

Corollary

$$\begin{aligned} \mathit{NLOGSPACE} &\subseteq \mathit{PTIME} \\ \mathit{co\text{-}NLOGSPACE} &\subseteq \mathit{PTIME} \\ \mathit{NPSPACE} &\subseteq \mathit{EXPTIME} \\ \mathit{co\text{-}NPSPACE} &\subseteq \mathit{EXPTIME} \end{aligned}$$

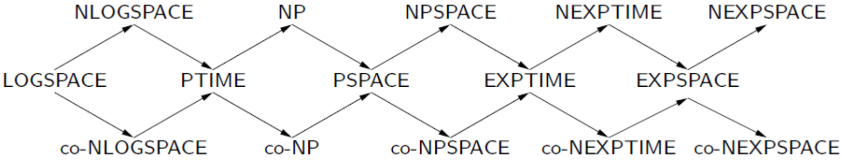
Complexity classes

As a corollary of the previous results we get:



Complexity classes

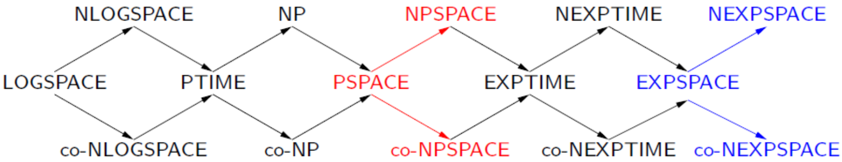
As a corollary of the previous results we get:



Can we reduce the figure even more?

Complexity classes

As a corollary of the previous results we get:



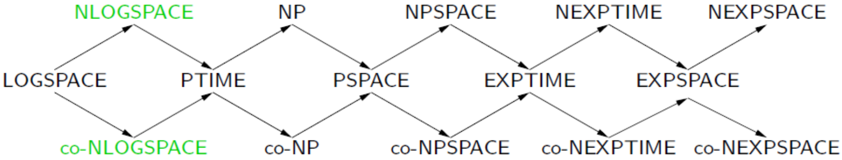
Savitch

Savitch

Can we reduce the figure even more?

Complexity classes

As a corollary of the previous results we get:



Immerman-Szelepcsényi

Can we reduce the figure even more?

We already did: Savitch's theorem

Theorem (Savitch)

$NSPACE(f(n)) \subseteq DSPACE(f(n)^2)$, for $f(n) \geq logn$.

Corollary

$$\begin{aligned} PSPACE &= NPSPACE \\ EXPSPACE &= NEXPSPACE \end{aligned}$$

Combined with previous results we get:

We already did: Savitch's theorem

Theorem (Savitch)

$NSPACE(f(n)) \subseteq DSPACE(f(n)^2)$, *for* $f(n) \geq logn$.

Corollary

$$\begin{aligned} PSPACE &= NPSPACE \\ EXPSPACE &= NEXPSPACE \end{aligned}$$

Combined with previous results we get:

- PSPACE = NPSPACE = co-NPSPACE

We already did: Savitch's theorem

Theorem (Savitch)

$NSPACE(f(n)) \subseteq DSPACE(f(n)^2)$, for $f(n) \geq logn$.

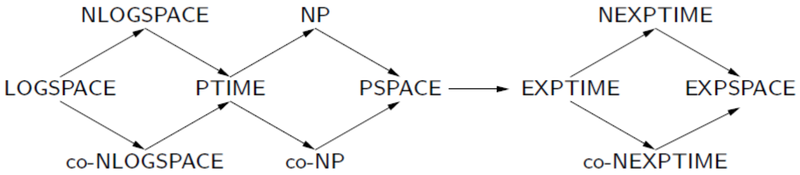
Corollary

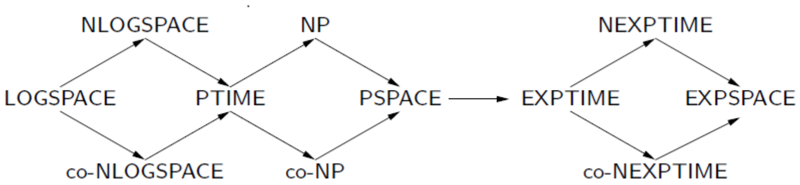
$$\begin{aligned} PSPACE &= NPSPACE \\ EXPSPACE &= NEXPSPACE \end{aligned}$$

Combined with previous results we get:

- PSPACE = NPSPACE = co-NPSPACE
- EXPSPACE = NEXPSPACE = co-NEXPSPACE

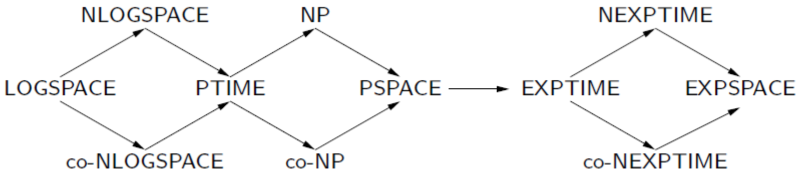
Complexity classes: what we get from Savitch





Next we resolve the question NLOGSPACE = co-NLOGSPACE?

Complexity classes: what we get from Savitch



Next we resolve the question NLOGSPACE = co-NLOGSPACE?

- The answer to this **does not** follow from Savitch's theorem.

Immerman-Szelepcsényi theorem

Theorem (Immerman-Szelepcsényi)

NLOGSPACE = *co-NLOGSPACE*

Proof: We show that $\overline{\text{PATH}}$ (the complement) is in NLOGSPACE.

Since PATH is NLOGSPACE-complete this is enough.

- Why?

Recall, in PATH we get G, s, t as input

We construct M that accepts iff **there is no path** from s to t

Immerman-Szelepcsényi theorem

For a graph G let $m = |G|$ (number of nodes).

Let c be the number of nodes reachable from s

If we have c we can efficiently determine if G has no path from s to t

Let us describe how this is done

Immerman-Szelepcsényi theorem: when I know c

$M =$ On input G, s, t and c do:

1. $d := 0$
2. For each node u in G:
3. Nondeterministically guess if u is reachable from s
4. If you guess yes, then guess a path of length m
5. If path does not reach u reject
6. If path contains t reject
7. $d++$ [Count the number of nodes verified to be reachable]
8. If $d \neq c$ reject, otherwise accept

Each guess is a new branch; we need only one to succeed

Immerman-Szelepcsényi theorem: computing c

Let A_i be the set of nodes reachable from s in **at most** i steps

Then $A_0 = \{s\}$ and $A_i \subseteq A_{i+1}$

Let $c_i = |A_i|$ (clearly $c = c_m$)

We compute c_{i+1} from c_i

The idea is the same as in M from previous slide (but nested twice)

To check that $v \in A_{i+1}$ we can use the previous algorithm knowing c_i

Immerman-Szelepcsényi theorem: computing c

Suppose we know c_i; to compute c_{i+1}:

1. For each node $v \in G$: (checking if $v \in A_{i+1}$)
2. $\quad d = 0$
3. $\quad$ For each $u \in G$:
4. $\quad\quad$ Nondeterministically guess if $u \in A_i$
5. $\quad\quad$ If you guess yes, then guess a path of length i
6. $\quad\quad$ Reject if the path does not reach u
7. $\quad\quad$ If path does reach u:
8. $\quad\quad\quad$ If (u, v) is an edge $c_{i+1} + +$ $\quad$ [(v, v) is an edge]
9. $\quad\quad$ $d + +$ [Count the number of nodes verified to be in A_i]
10. $\quad$ If $d \neq c_i$ reject, otherwise goto next v in (1)

Immerman-Szelepcsényi algorithm

$M =$ "On input $\langle G, s, t\rangle$:

1. Let $c_0 = 1$. $[\![\, A_0 = \{s\}$ has 1 node $]\!]$

2. For $i = 0$ to $m - 1$: $[\![$ compute c_{i+1} from c_i $]\!]$

3. Let $c_{i+1} = 1$. $[\![\, c_{i+1}$ counts nodes in A_{i+1} $]\!]$

4. For each node $v \neq s$ in G: $[\![$ check if $v \in A_{i+1}$ $]\!]$

5. Let $d = 0$. $[\![\, d$ re-counts A_i $]\!]$

6. For each node u in G: $[\![$ check if $u \in A_i$ $]\!]$

7. Nondeterministically either perform or skip these steps:

8. Nondeterministically follow a path of length at most i from s and *reject* if it doesn't end at u.

9. Increment d. $[\![$ verified that $u \in A_i$ $]\!]$

10. If (u, v) is an edge of G, increment c_{i+1} and go to Stage 5 with the next v. $[\![$ verified that $v \in A_{i+1}$ $]\!]$

11. If $d \neq c_i$, then *reject*. $[\![$ check whether found all A_i $]\!]$

12. Let $d = 0$. $[\![\, c_m$ now known; d re-counts A_m $]\!]$

13. For each node u in G: $[\![$ check if $u \in A_m$ $]\!]$

14. Nondeterministically either perform or skip these steps:

15. Nondeterministically follow a path of length at most m from s and *reject* if it doesn't end at u.

16. If $u = t$, then *reject*. $[\![$ found path from s to t $]\!]$

17. Increment d. $[\![$ verified that $u \in A_m$ $]\!]$

18. If $d \neq c_m$, then *reject*. $[\![$ check that found all of A_m $]\!]$
Otherwise, *accept*."

Immerman-Szelepcsényi theorem

What are the variables we have to keep track of?

- u and v (we always reuse space, so just the current one)
- c_i and c_{i+1} (and not all c_is)
- The counters d and i
- The pointer to the position in the path we are guessing

Each of these needs only LOGSPACE

(We also accept improper inputs)

$\square$

An easy generalisation

Corollary

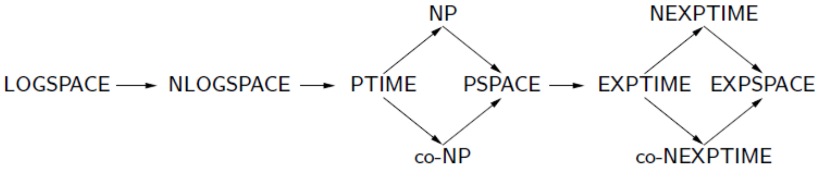
If $f(n) \geq logn$ is a proper complexity function, then

$$NSPACE(f(n)) \ = \ co\text{-}NSPACE(f(n))$$

To prove this use the Immerman-Szelepcsényi algorithm and run it over the configuration graph of the machine running in NSPACE($f(n)$).

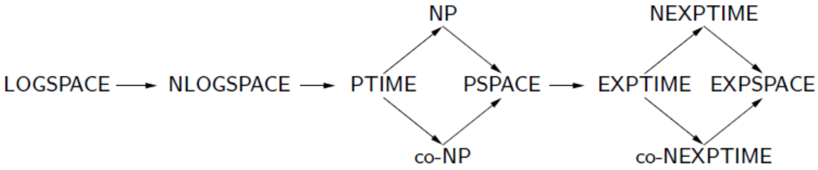
Immerman-Szelepcsényi theorem: consequences

The figure now looks like this:



Immerman-Szelepcsényi theorem: consequences

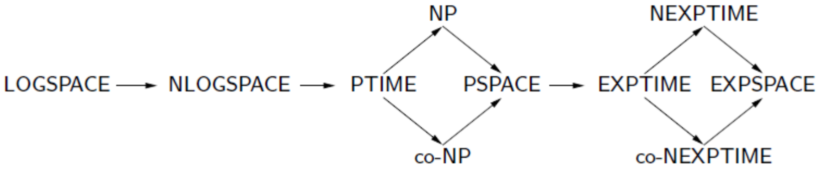
The figure now looks like this:



We still have many unknowns.

Immerman-Szelepcsényi theorem: consequences

The figure now looks like this:



We still have many unknowns.

- Which inclusions are proper?

Separating complexity classes: diagonalisation

To separate classes we will use the diagonalisation method

We will also need to define universal Turing machines

Which are abstractions of an actual computer

So let's start with this

Separating complexity classes: universal machine

We begin by considering deterministic space complexity classes.

- We will use the following assumptions.

Separating complexity classes: universal machine

We begin by considering deterministic space complexity classes.

- We will use the following assumptions.

First assumption

The input alphabet is always $\Sigma = \{0, 1\}$.

Separating complexity classes: universal machine

We begin by considering deterministic space complexity classes.

- We will use the following assumptions.

First assumption

The input alphabet is always $\Sigma = \{0, 1\}$.

Second assumption

We only consider TMs with a single work tape (and an input tape) with tape alphabet $\Gamma = \{0, 1, \texttt{B}, \vdash\}$.

Separating complexity classes: universal machine

We begin by considering deterministic space complexity classes.

- We will use the following assumptions.

First assumption

The input alphabet is always $\Sigma = \{0, 1\}$.

Second assumption

We only consider TMs with a single work tape (and an input tape) with tape alphabet $\Gamma = \{0, 1, \mathtt{B}, \vdash\}$.

Why can we assume this?

Separating complexity classes: universal machine

Theorem

For every deterministic TM M_1 with k work tapes there is a deterministic TM M_2 with one work tape such that:

- *The tape alphabet of M_2 is* $\Gamma = \{0, 1, \mathtt{B}, \vdash\}$
- $L(M_1) = L(M_2)$
- $s_{M_2}(n)$ *is* $O(s_{M_1}(n))$

Exercise

Prove this (we already saw how to remove tapes).

Separating complexity classes: universal machine

The idea of a universal Turing machine U:

$U =$ on input $\langle M, w \rangle$, with M a TM and w a word

- Simulate M on w
- If M accepts accepts; if M rejects reject

(Note: It can run forever)

Universal TM is a computer.

- How do we describe its input $\langle M, w \rangle$?
- How do we execute M over w?
- How much space are we using?

Coding a TM as a string over $\{0,1\}^*$

Let $M = (Q, \Sigma, \Gamma, q_{init}, q_{accept}, q_{reject}, \delta)$ be a single tape TM with input, where:

- $Q = \{q_1, \ldots, q_m\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, 1, \mathtt{B}, \vdash\}$
- $\delta : Q \times \Gamma \times \Gamma \to Q \times \{\leftarrow, \square, \rightarrow\} \times \Gamma \times \{\leftarrow, \square, \rightarrow\}$

(Here we use that first tape is input explicitly: no (re)writing)

The codification $\langle M \rangle$ of a TM M is a string over $\{0, 1\}$.

- We need this to pass it to U as input

Coding a TM as a string over $\{0,1\}^*$

For $M = (Q, \Sigma, \Gamma, q_{init}, q_{accept}, q_{reject}, \delta)$ with m states:

- We enumerate Q as $q_1, \ldots, q_m$
- The first state on the list (q_1) is the initial state q_{init}
- The last state (q_m) is q_{accept}
- The next to last state (q_{m-1}) is the reject state q_{reject}

We code each state of M as:

- The state q_i is represented by $cod(q_i) = 0^i$
- The initial state has code $cod(q_{init}) = 0$
- The accepting state has code $cod(q_{accept}) = 0^m$
- The rejecting state has code $cod(q_{reject}) = 0^{m-1}$

Coding a TM as a string over $\{0,1\}^*$

We use the following codes for symbols of M:

symbol	representation
0	0
1	00
B	000
$\vdash$	0000

symbol	representation
$\leftarrow$	0
$\square$	00
$\rightarrow$	000

If we now have a transition $t : \delta(q_i, a, b) = (q_j, D_1, c, D_2)$, we represent t as:

$$cod(t) = \underbrace{0^i}_{q_i} \; 1 \; \underbrace{0^{k_1}}_{a} \; 1 \; \underbrace{0^{k_2}}_{b} \; 1 \; \underbrace{0^j}_{q_j} \; 1 \; \underbrace{0^{d_1}}_{D_1} \; 1 \; \underbrace{0^l}_{c} \; 1 \; \underbrace{0^{d_2}}_{D_2}$$

with $i, j \in \{1, \ldots, m\}$; $\quad l, d_1, d_2 \in \{1, 2, 3\}$; $\quad k_1, k_2 \in \{1, 2, 3, 4\}$

Coding a TM as a string over $\{0,1\}^*$

For $M = (Q, \Sigma, \Gamma, q_{init}, q_{accept}, q_{reject}, \delta)$ with m states:

- $Q = \{q_1, \ldots, q_m\}$ and
- $\delta = (t_1, \ldots, t_k)$

We define the coding $\langle M \rangle$ of M as:

$$\langle M \rangle = \overbrace{0 \ldots 0}^{m-times} \; 111 \; cod(t_1) \; 11 \; cod(t_2) \; 11 \; \cdots \; 11 \; cod(t_k) \; 111$$

How do we know which states are initial, accepting, rejecting?

The universal machine U

We now define $\langle M, w \rangle = cod(M) \cdot w$

Our machine U uses five tapes:

- The first tape is the input tape containing $cod(M) \cdot w$
- The second tape will contain the (single) work tape of M
- The third tape contains the state M is in
- The fourth tape contains the position in w that M is reading
- The fifth tape contains the position of M on the work tape

The universal machine U: how does it work

To initialize M the universal machine U does:

1. Check that the input is $\langle M, w \rangle$ for some TM M (if not reject)
2. Write 0 on the third tape (initial state)
3. Write the first position of w on the fourth tape
4. Write 1 on the fifth tape

The universal machine U: how does it work

To simulate a step of M the universal machine U does:

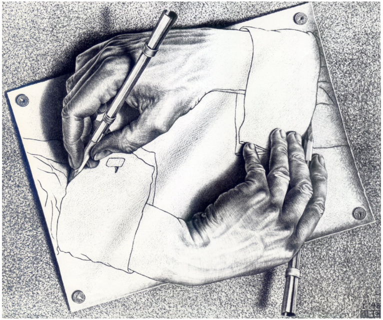
1. Look up the state q_i on third tape
2. Look up symbols a and b (the info is on tapes 4,5)
3. Look for a transition $\delta(q_i, a, b) = (q_j, D_1, c, D_2)$ on the input tape
4. Write q_j on the third tape (erase other stuff)
5. Change the second tape (replace b with c)
6. Change the content of tape $4/5$ according to D_1 and D_2 (make sure you stay on w)
7. If you see q_{accept} or q_{reject} do the same
8. Otherwise goto 1 again

It easily follows:

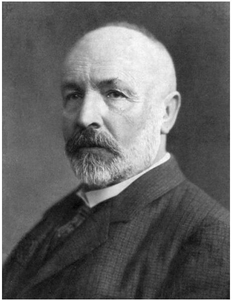
Theorem

The machine U accepts $\langle M, w\rangle$ iff M accepts w.

Diagonalisation: run U on $\langle U, cod(U)\rangle$ (but change U a bit)



Diagonalisation: slides stolen from Cristian Riveros



Technique invited by **Georg Cantor** to show that there is no bijection between $\mathbf{N}$ and its powerset:

$$2^{\mathbf{N}} = \{S \mid S \subseteq \mathbf{N}\}$$

Diagonalisation between $\mathbf{N}$ and $2^{\mathbf{N}}$

Assume (**to the contrary**) that f is a bijection from $\mathbf{N}$ to $2^{\mathbf{N}}$.

	0	1	2	3	4	5	6	7	$\cdots$
$f(0)$	1	1	0	1	0	0	1	1	
$f(1)$	0	0	1	1	1	0	0	1	
$f(2)$	1	1	1	1	0	0	0	0	
$f(3)$	1	0	1	0	0	1	0	1	
$f(4)$	0	0	1	1	0	0	1	0	$\cdots$
$f(5)$	1	1	0	1	0	1	1	1	
$f(6)$	1	0	0	0	0	0	1	0	
$f(7)$	1	0	0	1	0	1	1	1	
$\vdots$					$\vdots$				$\ddots$

The position (i, j) is equal to 1 iff $j \in f(i)$.

Each subset $S \in 2^{\mathbf{N}}$ is a row of the matrix

Diagonalisation between $\mathbf{N}$ and $2^{\mathbf{N}}$

Now consider the diagonal of the matrix:

	0	1	2	3	4	5	6	7	$\cdots$
$f(0)$	1	1	0	1	0	0	1	1	
$f(1)$	0	0	1	1	1	0	0	1	
$f(2)$	1	1	1	1	0	0	0	0	
$f(3)$	1	0	1	0	0	1	0	1	
$f(4)$	0	0	1	1	0	0	1	0	$\cdots$
$f(5)$	1	1	0	1	0	1	1	1	
$f(6)$	1	0	0	0	0	0	1	0	
$f(7)$	1	0	0	1	0	1	1	1	
$\vdots$					$\vdots$				$\ddots$

- The diagonal subset is equal to $D = \{i \in \mathbf{N} \mid i \in f(i)\}$.
- The **complement** of the diagonal is $\bar{D} = \{i \in \mathbf{N} \mid i \notin f(i)\}$.

Diagonalisation between $\mathbf{N}$ and $2^{\mathbf{N}}$

Definition (complement of the diagonal)

$$\bar{D} = \{i \in \mathbf{N} \mid i \notin f(i)\}$$

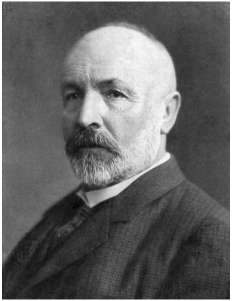
Does $\bar{D}$ appear as a row in the matrix?

NO, because $\bar{D}$ differs from $f(x)$ for all $x \in \mathbf{N}$.

$$x \in f(x) \quad \text{iff} \quad x \notin \bar{D}$$

Therefore such bijection f between $\mathbf{N}$ and $2^{\mathbf{N}}$ does not exist. $\square$

Diagonalisation between $\mathbf{N}$ and $2^{\mathbf{N}}$



Theorem

There is no bijection from $\mathbf{N}$ to $2^{\mathbf{N}}$.

"I see it, but I don't believe it!"

A letter from Cantor to Dedekind.

Diagonalisation of Turing machines

Consider the following problem:

$$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a Turing machine and } M \text{ accepts } w\}$$

Is this decidable?

Suppose that it is. Then there is H s.t:

$$H(\langle M, w\rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ is a TM accepting } w \\ \textit{reject} & \text{if } M \text{ is not a TM, or } M \text{ does not accept } w \end{cases}$$

Diagonalisation of Turing machines

Let us diagonalize now. Consider the machine:

$D =$ On input $\langle M \rangle$, for M a TM:

1. Run H on input $\langle M, \langle M \rangle \rangle$
2. If H accepts reject, if H rejects accept

Note that D is "reverse" of our universal machine!

That is:

$$D(\langle M \rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\ \textit{reject} & \text{if } M \text{ is not a TM, or } M \text{ accepts } \langle M \rangle \end{cases}$$

Diagonalisation of Turing machines

Now comes the kicker!

So what does D do with $\langle D \rangle$?

$$D(\langle D \rangle) = \begin{cases} \textit{accept} & \text{if } D \text{ does not accept } \langle D \rangle \\ \textit{reject} & \text{if } D \text{ accepts } \langle D \rangle \end{cases}$$

Contradiction, so no such H and D can not exist!

What was diagonal about that?

Clearly there are countably many TMs (strings over $\{0, 1\}$)

Let $M_1, M_2, \ldots$ be a listing of all of them

Each $\langle M_i \rangle$ is a string, so any M_j can be run with this input

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	accept	reject		accept	
M_2	accept	accept	accept	accept	
M_3					$\cdots$
M_4	accept	accept			
$\vdots$			$\vdots$		

What was diagonal about that?

With H we can fill in the table:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	accept	reject		accept	
M_2	accept	accept	accept	accept	
M_3					$\cdots$
M_4	accept	accept			
$\vdots$			$\vdots$		

What was diagonal about that?

With H we can fill in the table:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	accept	reject	reject	accept	
M_2	accept	accept	accept	accept	
M_3	reject	reject	reject	reject	$\cdots$
M_4	accept	accept	reject	reject	
$\vdots$			$\vdots$		

What was diagonal about that?

D looks what H does with $\langle M, \langle M \rangle \rangle$ as reverses it:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	accept	reject	reject	accept	
M_2	accept	accept	accept	accept	
M_3	reject	reject	reject	reject	$\cdots$
M_4	accept	accept	reject	reject	
$\vdots$			$\vdots$		

What was diagonal about that?

D looks what H does with $\langle M, \langle M \rangle \rangle$ as reverses it:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	reject	reject	reject	accept	
M_2	accept	accept	accept	accept	
M_3	reject	reject	reject	reject	$\cdots$
M_4	accept	accept	reject	reject	
$\vdots$			$\vdots$		

What was diagonal about that?

D looks what H does with $\langle M, \langle M \rangle \rangle$ as reverses it:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	reject	reject	reject	accept	
M_2	accept	reject	accept	accept	
M_3	reject	reject	reject	reject	$\cdots$
M_4	accept	accept	reject	reject	
$\vdots$			$\vdots$		

What was diagonal about that?

D looks what H does with $\langle M, \langle M \rangle \rangle$ as reverses it:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	reject	reject	reject	accept	
M_2	accept	reject	accept	accept	
M_3	reject	reject	accept	reject	$\cdots$
M_4	accept	accept	reject	reject	
$\vdots$			$\vdots$		

What was diagonal about that?

D looks what H does with $\langle M, \langle M \rangle \rangle$ as reverses it:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	reject	reject	reject	accept	
M_2	accept	reject	accept	accept	
M_3	reject	reject	accept	reject	$\cdots$
M_4	accept	accept	reject	accept	
$\vdots$			$\vdots$		

What was diagonal about that?

What about D?

D is also a TM, so it should be in the table.

But running D on $\langle D \rangle$ has a result different than running any M_i on $\langle M_i \rangle$!

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	reject	reject	reject	accept	
M_2	accept	reject	accept	accept	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$
D	accept	accept	$\cdots$	?	
$\vdots$			$\vdots$		

What was diagonal about that?

What about D?

D is also a TM, so it should be in the table.

But running D on $\langle D \rangle$ has a result different than running any M_i on $\langle M_i \rangle$!

$$
\begin{array}{c|cccccc}
 & \langle M_1 \rangle & \langle M_2 \rangle & \cdots & \langle D \rangle & \cdots \\
\hline
M_1 & \textcolor{red}{\text{reject}} & \text{reject} & \text{reject} & \text{accept} & \\
M_2 & \text{accept} & \textcolor{red}{\text{reject}} & \text{accept} & \text{accept} & \\
\vdots & \vdots & \vdots & \vdots & \vdots & \cdots \\
D & \text{accept} & \text{accept} & \cdots & \textcolor{red}{?} & \\
\vdots & & & \vdots & &
\end{array}
$$

Space hierarchy theorem

We will use diagonalisation to prove how to separate space complexity classes.

- The machine D above will play a key role.

Theorem (Space hierarchy theorem)

If $f(n) \geq logn$, then there is a language A decidable in space $O(f(n))$, but not in space $o(f(n))$.

Recall: $t_M(n) = o(f(n))$ if $\lim_{n\to\infty} \frac{t_M(n)}{f(n)} = 0$

Space hierarchy theorem: proof

Proof: We describe the language A using a machine deciding it.

The machine B for A is similar to D from the undecidability proof.

It is the same as our universal machine U, but acts opposite of it.

To ensure the lower bound we make sure that:

- For any M running in $o(f(n))$ space
- B differs from M on the input $\langle M \rangle$

Space hierarchy theorem: proof

High level description of B:

- Input is $\langle M \rangle$ for a TM M (recall our assumptions)
- If input is not $\langle M \rangle$ then B rejects
- Otherwise run M on $\langle M \rangle$ within $f(n)$ space
- If M halts (within $f(n)$ space) B accepts iff M rejects
- If M does not halt reject

Space hierarchy theorem: proof

Two possible issues:

1. What is M halts after $f(n)$ for small n?
   - This happens before $o(f(n))$ definition has kicked in
   - Solution: Inputs are $\langle M \rangle 01^*$ (recall what is $\langle M \rangle$)
   - So the problem is avoided on $\langle M \rangle 01^k$ for some k
2. What if M does not halt within ascribed space (ever)?
   - M is deterministic, so it would repeat a configuration
   - At most $2^{o(f(n))}$ time used by a $o(f(n))$ space machine
   - So just count up to $2^{f(n)}$ if you exceed reject

Space hierarchy theorem: proof

To formalise B we modify the universal machine U:

- Add one tape to U (to count up to $2^{f(n)}$)
- Add $\#$ to tape alphabet (to measure used space)

$B =$ On input w

1. For $n = |w|$ compute $f(n)$ (f is proper)
2. Mark off $f(n)$ space on each tape using $\#$
3. If $w \neq \langle M \rangle 01^*$ reject
4. "Do what U does" (run M on w), but also:
    - Count up to $2^{f(n)}$ on the last tape (reject if exceeded)
    - If you try to use a B reject (enforce $f(n)$ space)
    - If M accepts reject, if M rejects accept

Space hierarchy theorem: proof

B clearly uses $f(n)$ space, so $A \in \text{DSPACE}(f(n))$

Assume A is decidable in $o(f(n))$ space by some M

Then M runs in $g(n) = o(f(n))$, so for some n_0:

- If $n \geq n_0$ then $g(n) < f(n)$

Run B on $\langle M \rangle 01^{n_0}$ (stage 4 completes), so B is different than M

So A is not decidable in $o(f(n))$

$\square$

Space hierarchy theorem: consequences

This immediately gives us:

Corollary

For $f_1(n) = o(f_2(n))$, where $f_1, f_2 \geq logn$ are proper complexity functions we have:

$$DSPACE(f_1(n)) \subsetneq DSPACE(f_2(n)).$$

Space hierarchy theorem: consequences

Let's separate some classes.

Corollary

For every natural number $k \geq 1$:

$$\begin{aligned} DSPACE(n^k) &\subsetneq DSPACE(n^{k+1}) \\ DSPACE(2^{n^k}) &\subsetneq DSPACE(2^{n^{k+1}}) \end{aligned}$$

Exercise

Prove this.

Space hierarchy theorem: consequences

Corollary

$$LOGSPACE \subsetneq PSPACE \subsetneq EXPSPACE$$

Space hierarchy theorem: consequences

Corollary

$$LOGSPACE \subsetneq PSPACE \subsetneq EXPSPACE$$

Proof:

- $\text{LOGSPACE} \subseteq \text{DSPACE}(n) \subsetneq \text{DSPACE}(n^2) \subseteq \text{PSPACE}$
- $\text{PSPACE} \subseteq \text{DSPACE}(2^n) \subsetneq \text{DSPACE}(2^{n^2}) \subseteq \text{EXPSPACE}$

Space hierarchy theorem: consequences

We can also use the theorem to reason about nondeterministic space.

Corollary

$$\mathit{NLOGSPACE} \subsetneq \mathit{PSPACE}$$

Space hierarchy theorem: consequences

We can also use the theorem to reason about nondeterministic space.

Corollary

$$\mathit{NLOGSPACE} \subsetneq \mathit{PSPACE}$$

Proof:

$$\begin{aligned}\text{NLOGSPACE} \subseteq \text{NSPACE}(n) \subseteq \text{DSPACE}(n^2) \subsetneq \\ \text{DSPACE}(n^3) \subseteq \text{PSPACE}\end{aligned}$$

Separating complexity classes: Time hierarchy theorem

We can also use diagonalisation to separate time complexity classes.

Theorem (Time hierarchy theorem)

For every $t(n) \geq n \cdot log\ n$ there is a language A such that A is decidable in time $O(t(n))$, but not in time $o(t(n)/log\ t(n))$.

Separating complexity classes: Time hierarchy theorem

We can also use diagonalisation to separate time complexity classes.

Theorem (Time hierarchy theorem)

For every $t(n) \geq n \cdot log\ n$ there is a language A such that A is decidable in time $O(t(n))$, but not in time $o(t(n)/log\ t(n))$.

How do we prove this?

Separating complexity classes: Time hierarchy theorem

We can also use diagonalisation to separate time complexity classes.

Theorem (Time hierarchy theorem)

For every $t(n) \geq n \cdot \log n$ there is a language A such that A is decidable in time $O(t(n))$, but not in time $o(t(n)/\log t(n))$.

How do we prove this?

- What's with the $t(n)/logt(n)$ term?
- How long does simulation of M by U take?
- Hint: you have to count a lot.

Time hierarchy theorem: consequences

Using the time hierarchy theorem we can separate some complexity classes.

Corollary

For every natural number $k \geq 1$:

$$\begin{aligned} DTIME(n^k) &\subsetneq DTIME(n^{k+1}) \\ DTIME(2^{n^k}) &\subsetneq DTIME(2^{n^{k+1}}) \end{aligned}$$

Exercise

Prove the corollary.

Time hierarchy theorem: consequences

Corollary

$$PTIME \subsetneq EXPTIME$$

Time hierarchy theorem: consequences

Corollary

$$PTIME \subsetneq EXPTIME$$

Proof:

- $\mathrm{PTIME} \subseteq \mathrm{DTIME}(2^n) \subsetneq \mathrm{DTIME}(2^{n^2}) \subseteq \mathrm{EXPTIME}$

What is truly inefficient?

Regular expressions:

$$R \;:=\; \emptyset \;\mid\; \varepsilon \;\mid\; a \in \Sigma \;\mid\; R \cdot R \;\mid\; R + R \;\mid\; R^*$$

Regular expressions with exponents:

$$R \;:=\; \emptyset \;\mid\; \varepsilon \;\mid\; a \in \Sigma \;\mid\; R \cdot R \;\mid\; R + R \;\mid\; R^* \;\mid\; R^k (k \geq 1)$$

$$R^k = \underbrace{R \cdot R \cdots R}_{k-times}$$

Easy to see: same expressive power

What is truly inefficient?

Natural problems:

$$\mathsf{EQ}_{\mathsf{REX}} = \{\langle Q, R\rangle \mid Q, R \text{ are equivalent regular expressions}\}$$

Easy to see: This is PSPACE-complete

What is truly inefficient?

Natural problems:

$$\mathsf{EQ}_{\mathsf{REX}} = \{\langle Q, R\rangle \mid Q, R \text{ are equivalent regular expressions}\}$$

Easy to see: This is PSPACE-complete

$$\mathsf{EQ}_{\mathsf{REX}\uparrow} = \{\langle Q, R\rangle \mid Q, R \text{ are equivalent regular expressions with exponents}\}$$

Theorem

$EQ_{REX\uparrow}$ is EXPSPACE-complete.

So definitely not efficient.

EXPSPACE-completeness: the upper bound

Proof: We start with the upper bound.
The following decides if two NFAs are not equivalent:

$N =$ "On input $\langle N_1, N_2\rangle$, where N_1 and N_2 are NFAs:

1. Place a marker on each of the start states of N_1 and N_2.
2. Repeat $2^{q_1+q_2}$ times, where q_1 and q_2 are the numbers of states in N_1 and N_2:
3. Nondeterministically select an input symbol and change the positions of the markers on the states of N_1 and N_2 to simulate reading that symbol.
4. If at any point, a marker was placed on an accept state of one of the finite automata and not on any accept state of the other finite automaton, *accept*. Otherwise, *reject*."

Runs in NSPACE(n), so also in DSPACE(n^2).

EXPSPACE-completeness: the upper bound

To get the desired result we use the following:

E = On input $\langle R_1, R_2 \rangle$, where R_1, R_2 are regexp with exponentiation:

1. Convert R_1 and R_2 into equivalent regexp B_1 and B_2 that do not use exponentiation
2. Convert B_1 and B_2 into equivalent NFAs N_1, N_2
3. Run N from previous slide on $\langle N_1, N_2 \rangle$, output the opposite

Where is the exponential blowup?

EXPSPACE-completeness: the lower bound

For EXPSPACE-hardness take any A which is decidable in space 2^{n^k}, for some k (we'll disregard the constant) on a machine M

Let Q be the states of M, and Γ its tape alphabet

A *computation history* of M on w is:

$$C_1 \# C_2 \# C_3 \# \ldots \# C_k$$

Where:

- Each C_i is a configuration of M on w
- C_1 is the initial configuration
- C_{i+1} follows from C_i by transitions of M
- $\#$ does not appear in Γ

EXPSPACE-completeness: the lower bound

Accepting/rejecting computation history: according to C_k

Take $\Delta = Q \cup \Gamma \cup \{\#\}$

We will construct R_1, R_2, which are regexp with exponents such that:

$$R_1 \equiv R_2 \text{ iff } M \text{ accepts } w$$

Idea: M accepts w iff it has no rejecting computation histories of M on w

EXPSPACE-completeness: the lower bound

We take $R_1 = \Delta^*$

R_2 codes all strings that are not rejecting computation histories

Therefore $R_1 \equiv R_2$ iff M accepts w

Note that R_2 has to be polynomial!

$$R_2 = R_{bad-start} + R_{bad-reject} + R_{bad-window}$$

EXPSPACE-completeness: the lower bound

$R_{bad-start}$ are all strings not starting with the initial configuration of M on w

If $w = w_1 \cdots w_n$, then $C_1 =\vdash q_0 w_1 \cdots w_n \mathtt{B} \cdots \mathtt{B}\#$

$$R_{bad-start} = S_I + S_0 + S_1 + \cdots S_n + S_b + S_\#$$

Notation: Δ_{-a} is the union of all symbols in Δ, except for a

EXPSPACE-completeness: the lower bound

$$S_I = \Delta_{-\vdash}\Delta^*$$

$$S_0 = \vdash \cdot \Delta_{-q_0}\Delta^*$$

$$S_i = \vdash \cdot \Delta^i \Delta_{-w_i}\Delta^*, \text{ for } 1 \leq i \leq n$$

$$S_b = \vdash \cdot \Delta^{n+1}(\Delta + \varepsilon)^{2^{n^k}-n-2}\Delta_{-\mathtt{B}}\Delta^*$$

$$S_\# = \Delta^{2^{n^k}+1}\Delta_{-\#}\Delta^*$$

Similarly, $R_{bad-reject} = \Delta^*_{-q_{reject}}$

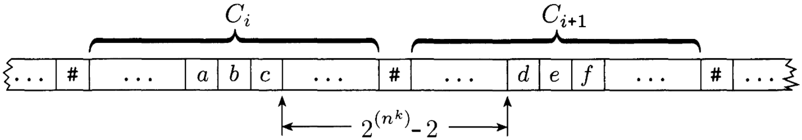
EXPSPACE-completeness: the lower bound

For $R_{bad-window}$ we use the notion of a window from Cook-Levin proof

I.e. C_i yields C_{i+1} if all three cell windows in doth are legal

$$R_{bad-window} = \bigcup_{\text{bad}(abc,def)} \Delta^* abc \Delta^{2^{n^k}-2} def \Delta^*$$

Here bad(abc, def) means that the top row abc can not yield def in the bottom row



On P vs NP

Is every problem in NP either NP-complete, or solvable in PTIME?

There are three possibilities:

1. There is $A \in \text{NP}$ that is not NP-complete, nor in PTIME
2. Every $A \in \text{NP}$ is either NP-complete, or in PTIME
3. $\text{PTIME} = \text{NP}$, so every $A \in \text{NP}$ is complete

On P vs NP: Ladner's theorem

Theorem (Ladner)

If PTIME $\neq$ NP, then there exists a language A such that:

- *A is in NP,*
- *A is* **not** *NP-complete; and*
- *A is* **not** *solvable in PTIME.*

Proof: We will use many ideas here (but it's really easy).

Basic idea: Define A by removing elements from SAT

Assume $\{0, 1\}$ as the input alphabet and usual stuff

Ladner's theorem: proof

Notation: M a TM and x a word, then $M(x)$ is accept/reject

Let $\mathcal{M}$ be the set of all (deterministic) TMs (countable)

The set $\mathcal{M} \times \mathbb{N}$ is also countable

Define a TM M_i as follows ($i \in \mathbb{N}$):

- The ith element of $\mathcal{M} \times \mathbb{N}$ is (M, j)
- $M_i(x)$ runs $M(x)$ for at most $|x|^j$ steps

The sequence $M_1, M_2, \ldots$ enumerates all poly-time DTMs

- (Single tape deciders that run in poly-time)

Ladner's theorem: proof

Similarly we can enumerate all poly-time reductions $F_1, F_2, \ldots$

- These compute functions, so have an output tape

Our language A has the following two properties:

- $A \notin$ PTIME
- A is not NP-complete

The first one means that $A \neq L(M_i)$ for all i

The second one that F_i is not a reduction from SAT to A

Ladner's theorem: double diagonalisation

To achieve the two properties we have two (infinite) set of requirements that need to be fulfilled:

1. $NotPol_i$: $\quad A \neq L(M_i)$
2. $NotCompl_i$: $\quad \exists x$ s.t.
   - $x \in \text{SAT}$ and $F_i(x) \notin A$; or
   - $x \notin \text{SAT}$ and $F_i(x) \in A$

This is done by (double) delayed diagonalisation

Ladner's theorem: proof

$$A = \{x \mid x \in \mathsf{SAT} \wedge f(|x|) \text{ is even}\}$$

The proof is in defining f

f will be defined by the machine M_f computing it

Start with $f(0) = f(1) = 2$

Let M_{SAT} be a DTM for SAT (exponential one)

Ladner's theorem: proof

On input $1^n (n > 1)$ our machine M_f works in two stages:

First stage: Do the following for n steps:

- Compute $f(0), f(1), \ldots$ until you run out of time
- Assume you stopped with $f(x) = k$
- The output will be k or $k+1$ depending on

Second stage: Do the following for n steps:

- If $k = 2i$ try to make sure that $NotPol_i$ holds
- If $k = 2i - 1$ try to make sure that $NotCompl_i$ holds

Ladner's theorem: second stage

Second stage: Do the following for n steps:

If $k = 2i$ try to make sure that $NotPol_i$ holds:

- We need $z \in \{0,1\}^*$ s.t. $M_i(z)$ is wrong
- (That is $M_i(z)$ accepts and $z \notin A$, or the opposite)
- For all z in lexicographical order compute:
  - $M_i(z)$, $M_{\mathsf{SAT}}(z)$ and $f(|z|)$
  - If z that confirms $NotPol_i$ is found (in n steps) output $k+1$
  - Otherwise output k

Ladner's theorem: second stage

Second stage: Do the following for n steps:

If $k = 2i - 1$ try to make sure that $NotCompl_i$ holds:

- We need $z \in \{0,1\}^*$ s.t. $F_i(z)$ is wrong
- (That is $z \in \mathrm{SAT}$ and $F_i(z) \notin A$, or the opposite)
- For all z in lexicographical order compute:
  - $F_i(z)$, $M_{\mathrm{SAT}}(z), M_{\mathrm{SAT}}(F_i(z))$ and $f(|F_i(z)|)$
  - If z that confirms $NotCompl_i$ is found (in n steps) output $k+1$
  - Otherwise output k

Ladner's theorem: why it works

By definition M_f is polynomial (also $f(n+1) \geq f(n)$)

Since $A = \{x \mid x \in \mathsf{SAT} \wedge f(|x|) \text{ is even}\}$

We get that $A \in \mathsf{NP}$:

- Compute $f(|x|)$ which is poly; if result is even:
- Do a guess and check for $x \in \mathsf{SAT}$

Ladner's theorem: why it works

We claim that $A \notin \text{PTIME}$

Assume the contrary, i.e. $A = L(M_i)$ for some i

Then second stage of M_f with $k = 2i$ never finds the needed z

But then f never reaches the value $k + 1$

So f is odd for only finitely many n

So A and SAT are the same apart for finitely many elements

So SAT $\in$ PTIME, but we assumed PTIME $\neq$ NP contradiction

Ladner's theorem: the end

We claim that A is not NP-complete:

If yes, then some F_i reduces SAT to A

As before, M_f with $k = 2i - 1$ never moves to $k + 1$

So f is even for only finitely many n

So A is finite set, thus $A \in$ PTIME contradiction

$\square$

Can we prove P vs NP using diagonalisation?

For space/time hierarchy we can diagonalise against all $o(f(n))$ machines

It is not clear how to do this on a single branch in NP computation

In fact, separating P and NP using diagonalisation is not likely

Why?

- Enter Oracle machines (not the company)

Oracle machines

Idea: machine with access to a black-box

The black box decides a language O

Oracle machine can ask if a $w \in O$:

- Black box answers in **one step**

Oracle machines

Definition

A deterministic TM with input tape and an oracle for $A \subseteq \Sigma^*$:
$M^A = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

- Q a finite set of states with $q_?, q_{yes}, q_{no} \in Q$
- Σ a finite alphabet with $\texttt{B} \notin \Sigma$
- Γ a finite alphabet with $\Sigma \cup \{\texttt{B}, \vdash\} \subseteq \Gamma$
- $q_0, q_{accept}, q_{reject} \in Q$ as before
- δ is a partial function:

$$\delta \; : \; Q \times \Gamma \times \Gamma^2 \to Q \times \{\leftarrow, \square, \rightarrow\} \times \Gamma^2 \times \{\leftarrow, \square, \rightarrow\}^2$$

  The third tape is the **query tape**

A is called the **oracle**

How do oracle machines work?

Easy to extend the previous definition to multiple work tapes and to non-determinism.

An oracle machine M^A works as an ordinary TM, but when it enters the state $q_?$:

- The query tape has: $w\text{BB}\cdots$, for some $w \in \Sigma^*$
- M uses the oracle for A, and its next state is q_{yes} or q_{no}
  - The state is q_{yes} iff $w \in A$

The execution time is defined as for ordinary TMs

- **A query to the oracle counts as one step**

Greeting from Delphi: $\text{PTIME}^{\text{SAT}}$

Oracle complexity classes: PTIME^A and NP^A (for now):

- E.g. $L \in \text{PTIME}^A$ if there is a poly-time TM M s.t. $w \in L$ iff M^A accepts w

$\text{PTIME}^{\text{SAT}}$ decides all problems in NP:

- If $B \in \text{NP}$ our machine M is just a reduction f from B to SAT
- On input w it computes $f(w)$
- And asks this to the oracle
- It accepts iff $f(w) \in \text{SAT}$ iff $w \in B$

More Greek postcards: $\mathrm{PTIME}^{\mathrm{TQBF}}$

Theorem

$$PTIME^{TQBF} = NP^{TQBF}.$$

- First, $\mathrm{NP}^{TQBF} \subseteq \mathrm{NPSPACE}$:
  - *TQBF* solvable in PSPACE, so every oracle call is unravelled like this
- Second, NPSPACE = PSPACE (Savitch)
- Third, $\mathrm{PSPACE} \subseteq \mathrm{PTIME}^{TQBF}$
  - Again, we just use the reduction from any PSPACE problem to TQBF

Oracle = relativisation

For any two complexity classes $\mathcal{A}, \mathcal{B}$ we can define the oracle version $\mathcal{A}^O$ and $\mathcal{B}^O$, for any oracle O

If some result holds for $\mathcal{A}, \mathcal{B}$ does it also hold for $\mathcal{A}^O$ and $\mathcal{B}^O$?

If so the result **relativises**

Limits of diagonalisation

Diagonalisation for TMs:

1. TMs can be represented by strings (we can enumerate them)
2. **One TM can simulate another one** (with polynomial overhead)

Results proved via diagonalisation relativise

Proof: Just plug in the oracle and nothing changes

Therefore if we show equal/different using diagonalisation, we can just plug in the oracle and the proof still holds

Limits of diagonalisation: Baker, Gill, Solovay

Theorem (Baker, Gill, Solovay 1975)

There exist oracles A and B such that:

1. $PTIME^A = NP^A$ *and;*
2. $PTIME^B \neq NP^B$.

Proof: We already proved the first claim ($A = TQBF$)

For the second claim we use diagonalisation (Oh the irony!)

We construct B as follows

Limits of diagonalisation: Baker, Gill, Solovay

For any oracle X define

$$U_X = \{1^n \mid \exists x \in X \text{ s.t. } |x| = n\}$$

Clearly $U_X \in \mathsf{NP}^X$ for any X:

- Just guess a string of equal length and ask the oracle

Let $M_1, M_2, \ldots$ be an enumeration of all oracle poly-time TMs

- Note that these do not depend on the oracle

We construct B such that:

- $U_B \neq L(M_i^B)$, for all i (the diagonal)
- I.e. $U_B \notin \mathsf{PTIME}^B$, but $U_B \in \mathsf{NP}^B$

Limits of diagonalisation: Baker, Gill, Solovay

$M_1, M_2, \ldots$ our enumeration of poly-time oracle machines

- Wlog. assume that M_i runs in time n^i, and that $\Sigma = \{0, 1\}$

Start with $i = 0$ and $B = \emptyset$

We construct B in stages such that:

- Stage i ensures that M_i^B doesn't decide U_B
- Each stage puts a finite amount of strings into B
- We begin with stage 1

Limits of diagonalisation: Baker, Gill, Solovay

stage i: We know only finitely many elements of B

Chose n s.t.

- $2^n > n^i$ (recall M_i runs in n^i time)
- n is larger then length of anything already in B (*or decided to be outside* B)

Idea: extend B s.t. M_i^B accepts 1^n iff $1^n \notin U_B$

Construction: run M_i^B on 1^n and reply to oracle queries as follows:

- If it asks about y and we already know $y \in B$ reply YES
- If we still don't know about y reply NO

Limits of diagonalisation: Baker, Gill, Solovay

stage i: continued

Let $w_1, \ldots, w_k$ be all strings that M_i used to query the oracle (on our input 1^n)

We know $k < n^i < 2^n$, so there is $w_0 \in \{0,1\}^n$ different from all $w_1, \ldots, w_k$

Expand B:

- If M_i accepts 1^n, then no string of length n is in B
- If M_i rejects 1^n, then put w_0 in B

Move to $i + 1$

In the end lengths no considered: outside of B

Limits of diagonalisation: Baker, Gill, Solovay

Why does this work?

In stage i:

- We chose our n
- M_i^B is polynomial, so it can't query all $w \in \{0,1\}^n$
- So it queried only $w_1, \ldots, w_k$, with $k < 2^n$
- We answered NO to all queries of length n (so they are not in B)
- If it accepted, we make no string of length n inside B
- If it rejected, we put one it didn't ask for inside B
- Therefore $L(M_i^B) \neq U_B$

$\square$