Computational Complexity; slides 6, HT 2019
Space complexity

Prof. Paul W. Goldberg (Dept. of Computer Science,
University of Oxford)

HT 2019

Road map

I mentioned classes like LOGSPACE (usually called L), SPACE($f(n)$) etc. How do they relate to each other, and time complexity classes?

Next: Various inclusions can be proved, some more easy than others; let's begin with "low-hanging fruit"...

e.g., I have noted: TIME($f(n)$) is a subset of SPACE($f(n)$) (easy!)

We will see e.g. L is a proper subset of PSPACE, although it's unknown how they relate to various intermediate classes, e.g. P, NP

Various interesting problems are complete for PSPACE, EXPTIME, and some of the others.

Space Complexity

So far, we have measured the complexity of problems in terms of the time required to solve them.

Alternatively, we can measure the space/memory required to compute a solution.

Important difference: space can be re-used

Space Complexity

So far, we have measured the complexity of problems in terms of the time required to solve them.

Alternatively, we can measure the space/memory required to compute a solution.

Important difference: space can be re-used

Convention: In this section we will be using Turing machines with a designated read only input tape. So, "logarithmic space" becomes meaningful.

Space Complexity

Definition. Let $\mathcal{M}$ be a Turing acceptor with designated input tape.

SPACE$_{\mathcal{M}}(w)$: the maximum number of non-blank cells of the work tapes during the computation of $\mathcal{M}$ on input $w \in \Sigma^*$.

Space Complexity

Definition. Let $\mathcal{M}$ be a Turing acceptor with designated input tape.

$\text{SPACE}_{\mathcal{M}}(w)$: the maximum number of non-blank cells of the work tapes during the computation of $\mathcal{M}$ on input $w \in \Sigma^*$.

Definition. Let $\mathcal{M}$ be a Turing accepter and $S : \mathbb{N} \to \mathbb{N}$ a monotone growing function.

$\mathcal{M}$ is <u>S-space bounded</u> if it halts on every input $w \in \Sigma^*$ and

$$\text{SPACE}_{\mathcal{M}}(w) \leq S(|w|).$$

1. DSPACE(S) is the class of languages $\mathcal{L}$ for which there is an S-space bounded k-tape deterministic Turing accepter deciding $\mathcal{L}$ for some $k \geq 1$.

2. NSPACE(S) is the class of languages $\mathcal{L}$ for which there is an S-space bounded non-deterministic k-tape Turing accepter deciding $\mathcal{L}$ for some $k \geq 1$.

Space Complexity Classes

- Deterministic Classes:
  - LOGSPACE $:= \bigcup_{d \in \mathbb{N}} \mathrm{DSPACE}(d \log n)$
  - PSPACE $:= \bigcup_{d \in \mathbb{N}} \mathrm{DSPACE}(n^d)$
  - EXPSPACE $:= \bigcup_{d \in \mathbb{N}} \mathrm{DSPACE}(2^{n^d})$
- Non-Deterministic versions: NLOGSPACE etc

Straightforward observation:

$$
\begin{array}{ccccc}
\mathrm{LOGSPACE} & \subseteq & \mathrm{PSPACE} & \subseteq & \mathrm{EXPSPACE} \\
\cap\!\!| & & \cap\!\!| & & \cap\!\!| \\
\mathrm{NLOGSPACE} & \subseteq & \mathrm{NPSPACE} & \subseteq & \mathrm{NEXPSPACE}
\end{array}
$$

Elementary relationships between time and space

Easy observation:

For all functions $f : \mathbb{N} \to \mathbb{N}$:

$$\mathsf{DTIME}(f) \subseteq \mathsf{DSPACE}(f)$$
$$\mathsf{NTIME}(f) \subseteq \mathsf{NSPACE}(f)$$

A bit harder:

For all monotone growing functions $f : \mathbb{N} \to \mathbb{N}$:

$$\mathsf{DSPACE}(f) \subseteq \mathsf{DTIME}(2^{\mathcal{O}(f)})$$
$$\mathsf{NSPACE}(f) \subseteq \mathsf{DTIME}(2^{\mathcal{O}(f)})$$

Elementary relationships between time and space

Easy observation:

For all functions $f : \mathbb{N} \to \mathbb{N}$:

$$\begin{aligned} \mathrm{DTIME}(f) &\subseteq \mathrm{DSPACE}(f) \\ \mathrm{NTIME}(f) &\subseteq \mathrm{NSPACE}(f) \end{aligned}$$

A bit harder:

For all monotone growing functions $f : \mathbb{N} \to \mathbb{N}$:

$$\begin{aligned} \mathrm{DSPACE}(f) &\subseteq \mathrm{DTIME}(2^{\mathcal{O}(f)}) \\ \mathrm{NSPACE}(f) &\subseteq \mathrm{DTIME}(2^{\mathcal{O}(f)}) \end{aligned}$$

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

- Build the configuration graph $\rightsquigarrow$ time $2^{\mathcal{O}(f(n))}$
- Find a path from the start to an accepting stop configuration. $\rightsquigarrow$ time $2^{\mathcal{O}(f(n))}$

Number of Possible Configurations

Let $\mathcal{M} := (Q, \Sigma, \Gamma, q_0, \Delta, F_a, F_r)$ be a 1-tape Turing accepter.
(plus input tape)

Recall: Configuration of $\mathcal{M}$ is a triple (q, p, x) where

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

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

If $\mathcal{M}$ is $f(n)$-space bounded we can assume that $p \leq f(n)$ and $|x| \leq f(n)$

Hence, there are at most

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

different configurations on inputs of length n.

Configuration Graphs

Let $\mathcal{M} := (Q, \Sigma, \Gamma, q_0, \Delta, F_a, F_r)$ be a 1-tape Turing accepter.
$f(n)$ space bounded

Configuration graph $\mathcal{G}(\mathcal{M}, w)$ of $\mathcal{M}$ on input w:
Directed graph with

Vertices: All possible configurations of $\mathcal{M}$ up to length $f(|w|)$

Edges: Edge $(C_1, C_2) \in E(\mathcal{G}(\mathcal{M}, w))$, if $C_1 \vdash_{\mathcal{M}} C_2$

A computation of $\mathcal{M}$ on input w corresponds to a path in $\mathcal{G}(\mathcal{M}, w)$ from the start configuration to a stop configuration.

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

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

Recall: L commonly denotes LOGSPACE; NL=NLOGSPACE

$$
\begin{array}{ccccccccc}
\text{L} & & & & & & & & \\
\rotatebox{-90}{$\subseteq$} & & & & & & & & \\
\text{NL} & \subseteq & \text{P} & \subseteq & \text{PSPACE} & & & & \\
 & & \rotatebox{-90}{$\subseteq$} & & \rotatebox{-90}{$\subseteq$} & & & & \\
 & & \text{NP} & \subseteq & \text{NPSPACE} & \subseteq & \text{EXPTIME} & \subseteq & \text{EXPSPACE} \\
 & & & & & & \rotatebox{-90}{$\subseteq$} & & \rotatebox{-90}{$\subseteq$} \\
 & & & & & & \text{NEXPTIME} & \subseteq & \text{NEXPSPACE}
\end{array}
$$

Simulating non-deterministic computations with limited space

Easy observation: SAT can be solved in linear space

Just try every possible assignment, one after another, reusing space.

Simulating non-deterministic computations with limited space

Easy observation: SAT can be solved in linear space

Just try every possible assignment, one after another, reusing space.

Consequence: NP $\subseteq$ PSPACE
similarly, NEXPTIME is a subset of EXPSPACE

Generally, non-deterministic time $f(n)$ allows $O(f(n))$ non-deterministic "guesses"; try them all one-by-one, in lexicographic order, over-writing previous attempts.

$$
\begin{array}{ccccccccc}
\mathsf{L} & & & & & & & & \\
\rotatebox[origin=c]{-90}{$\subseteq$} & & & & & & & & \\
\mathsf{NL} & \subseteq & \mathsf{P} & & \mathsf{PSPACE} & & & & \\
& & \rotatebox[origin=c]{-90}{$\subseteq$} & \rotatebox[origin=c]{-45}{$\subsetneq$} & \rotatebox[origin=c]{-90}{$\subseteq$} & & & & \\
& & \mathsf{NP} & & \mathsf{NPSPACE} & \subseteq & \mathsf{EXPTIME} & & \mathsf{EXPSPACE} \\
& & & & & & \rotatebox[origin=c]{-90}{$\subseteq$} & \rotatebox[origin=c]{-45}{$\subsetneq$} & \rotatebox[origin=c]{-90}{$\subseteq$} \\
& & & & & & \mathsf{NEXPTIME} & & \mathsf{NEXPSPACE}
\end{array}
$$

By the *time hierarchy theorem* (coming up next), P $\subsetneq$ EXPTIME, NP $\subsetneq$ NEXPTIME

By the *space hierarchy theorem*, NL $\subsetneq$ PSPACE, PSPACE $\subsetneq$ EXPSPACE.

Time Hierarchy theorem

proper complexity function f: roughly, an increasing function that can be computed by a TM in time $f(n) + n$

For $f(n) \geq n$ a proper complexity function, we have

$$\text{TIME}(f(n)) \text{ is a proper subset of } \text{TIME}((f(2n+1))^3).$$

It follows that P is a proper subset of EXPTIME.

Proof sketch: consider "time-bounded halting language"

$$H_f := \{\langle M, w\rangle \ : \ M \text{ accepts } w \text{ after } \leq f(|w|) \text{ steps}\}$$

H_f belongs to $\text{TIME}((f(n))^3)$: construct a universal TM that uses "quadratic overhead" to simulate a step of M. (The theorem can be strengthened by using a more economical UTM, but as stated it's good enough for P$\subsetneq$EXPTIME.)

Next point: $H_f \notin \text{TIME}(f(\lfloor \frac{n}{2} \rfloor))$.

Time Hierarchy theorem

Reminder:

$$H_f := \{\langle M, w\rangle \ : \ M \text{ accepts } w \text{ after } \leq f(|w|) \text{ steps}\}$$

To prove $H_f \notin \text{TIME}(f(\lfloor \frac{n}{2} \rfloor))$:

- Suppose M_{H_f} decides H_f in time $f(\lfloor \frac{n}{2} \rfloor)$.
- Define "diagonalising" machine:
  $D_f(M) \ : \ \text{if } M_{H_f}(\langle M, M\rangle) =$ "yes" then "no" else "yes"
- Does D_f accept its own description? Contradiction!

Corollary

P is a proper subset of EXPTIME

Next: PSPACE-completeness and Quantified Boolean Formulae

From polynomial space to linear space

Generic PSPACE-complete problem P_1; fix p, a polynomial

Input: $\langle M, w \rangle$
Question: Does M accept w in space $O(p(|w|))$?

Linear space version P_2:

Input: $\langle M, w \rangle$
Question: Does M accept w in space $O(|w|)$?

From polynomial space to linear space

Generic PSPACE-complete problem P_1; fix p, a polynomial

Input: $\langle M, w\rangle$
Question: Does M accept w in space $O(p(|w|))$?

Linear space version P_2:

Input: $\langle M, w\rangle$
Question: Does M accept w in space $O(|w|)$?

Easy theorem: $P_1 \leq_p P_2$.
To reduce P_1 to P_2,

$$\langle M, w\rangle \mapsto \langle M, w\mathtt{b}^{p(|w|)}\rangle$$

where b denotes the blank symbol. That is, we can "pad" the original input to give ourselves more space.

Let M be an NPSPACE TM of interest; want to know whether M can accept w within $2^{p(n)}$ steps.

Proof idea: predicate $\text{reachable}(C, C', i)$ is satisfied by configurations C, C' and integer i, provided C' is reachable from C within 2^i transitions (w.r.t M).

Note: $\text{reachable}(C, C', i)$ is satisfied provided there exists C'' such that
$\text{reachable}(C, C'', i-1)$ and $\text{reachable}(C'', C', i-1)$

To check $\text{reachable}(C_{init}, C_{accept}, p(n))$, try for all configs C'':
$\text{reachable}(C_{init}, C'', p(n)-1)$ and $\text{reachable}(C'', C_{accept}, p(n)-1)$

Which themselves are checked recursively. Depth of recursion is $p(n)$, need to remember at most $p(n)$ configs at any time. We may assume C_{accept} is unique.

More generally:

Theorem. (Savitch 1970)

For all (space-constructible) $S : \mathbb{N} \to \mathbb{N}$ such that $S(n) \geq \log n$,

$$\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DSPACE}(S(n)^2).$$

In particular: PSPACE = NPSPACE

EXPSPACE = NEXPSPACE

Quantified Boolean Formulae: Syntax

A Quantified Boolean Formula is a formula of the form

$$Q_1X_1 \dots Q_nX_n\varphi(X_1, \dots, X_n)$$

where

- the Q_i are quantifiers $\exists$ or $\forall$
- φ is a CNF formula in the variables $X_1, \dots, X_n$ and atoms 0 and 1

Example

$\exists X_1 \forall X_2 \exists X_3 \forall X_4 \forall X_5 \Big((X_1 \vee 0 \vee \neg X_5) \ \wedge \ (\neg X_2 \vee 1 \vee \neg X_5) \ \wedge \ (X_2 \vee X_3 \vee X_4) \Big)$

Quantified Boolean Formulae: Semantics

Definition. A quantified boolean formula φ is true if

- φ does not contain any quantifiers (and hence no variables) and it evaluates to *true*.
- $\varphi := \exists X \psi$ and $\psi[X \mapsto 0]$ or $\psi[X \mapsto 1]$ is true.
- $\varphi := \forall X \psi$ and both $\psi[X \mapsto 0]$ and $\psi[X \mapsto 1]$ are true.

Here $\psi[X \mapsto 1]$ is the formula obtained from ψ by replacing each occurrence of a literal X by 1 and $\neg X$ by 0. Analogously for $\psi[X \mapsto 0]$.

Quantified Boolean Formulae: Semantics

Definition. A quantified boolean formula φ is true if

- φ does not contain any quantifiers (and hence no variables) and it evaluates to *true*.
- $\varphi := \exists X \psi$ and $\psi[X \mapsto 0]$ or $\psi[X \mapsto 1]$ is true.
- $\varphi := \forall X \psi$ and both $\psi[X \mapsto 0]$ and $\psi[X \mapsto 1]$ are true.

Here $\psi[X \mapsto 1]$ is the formula obtained from ψ by replacing each occurrence of a literal X by 1 and $\neg X$ by 0. Analogously for $\psi[X \mapsto 0]$.

Example

$$\forall X_1 \; \forall X_2 \; \exists X_3 \quad \Big((\neg X_1 \vee \neg X_2 \vee X_3) \quad \wedge \quad (\neg X_1 \vee X_2 \vee \neg X_3) \Big)$$

Definition. A quantified boolean formula φ is true if

- φ does not contain any quantifiers (and hence no variables) and it evaluates to *true*.
- $\varphi := \exists X \psi$ and $\psi[X \mapsto 0]$ or $\psi[X \mapsto 1]$ is true.
- $\varphi := \forall X \psi$ and both $\psi[X \mapsto 0]$ and $\psi[X \mapsto 1]$ are true.

Here $\psi[X \mapsto 1]$ is the formula obtained from ψ by replacing each occurrence of a literal X by 1 and $\neg X$ by 0. Analogously for $\psi[X \mapsto 0]$.

Example

$\forall X_1 \; \forall X_2 \; \exists X_3 \quad \Big((\neg X_1 \vee \neg X_2 \vee X_3) \quad \wedge \quad (\neg X_1 \vee X_2 \vee \neg X_3) \Big)$ is true.

$\exists Y_1 \; \forall Y_2 \; \forall Y_3 \quad \Big((Y_1 \vee \neg Y_2 \vee \neg Y_3) \quad \wedge \quad (\neg Y_1 \vee Y_2 \vee \neg Y_3) \Big)$

Quantified Boolean Formulae: Semantics

Definition. A quantified boolean formula φ is true if

- φ does not contain any quantifiers (and hence no variables) and it evaluates to *true*.
- $\varphi := \exists X \psi$ and $\psi[X \mapsto 0]$ or $\psi[X \mapsto 1]$ is true.
- $\varphi := \forall X \psi$ and both $\psi[X \mapsto 0]$ and $\psi[X \mapsto 1]$ are true.

Here $\psi[X \mapsto 1]$ is the formula obtained from ψ by replacing each occurrence of a literal X by 1 and $\neg X$ by 0. Analogously for $\psi[X \mapsto 0]$.

Example

$\forall X_1 \; \forall X_2 \; \exists X_3 \quad \Big((\neg X_1 \vee \neg X_2 \vee X_3) \quad \wedge \quad (\neg X_1 \vee X_2 \vee \neg X_3) \Big)$ is true.

$\exists Y_1 \; \forall Y_2 \; \forall Y_3 \quad \Big((Y_1 \vee \neg Y_2 \vee \neg Y_3) \quad \wedge \quad (\neg Y_1 \vee Y_2 \vee \neg Y_3) \Big)$ is false.

Quantified Boolean Formulae

Consider the following problem:

QBF	
Input:	A QBF formula φ.
Problem:	Is φ true?

Observation: For any propositional formula φ:

φ is satisfiable if, and only if, $\exists X_1 \ldots \exists X_n \varphi$ is true.

$X_1, \ldots, X_n$: Variables occurring in φ

Consequence: QBF is NP-hard.

Theorem: QBF is in PSPACE

Proof: Given $\varphi := Q_1X_1 \; \ldots Q_nX_n\psi$, letting $m := |\psi|$

Eval-QBF(φ):

if $n = 0$ Accept if ψ evaluates to true. Reject otherwise.

if $\varphi := \exists X \psi'$
construct $\varphi_1 := \psi'[X \mapsto 1]$
if Eval-QBF(φ_1) evaluates to true, accept.
else construct $\varphi_0 := \psi'[X \mapsto 0]$ (reuse space in Eval-QBF(φ_1))
return Eval-QBF(φ_0)

if $\varphi := \forall X \psi'$
construct $\varphi_1 := \psi'[X \mapsto 1]$
if Eval-QBF(φ_1) evaluates to false, reject.
else construct $\varphi_0 := \psi'[X \mapsto 0]$ (reuse space in Eval-QBF(φ_1))
return Eval-QBF(φ_0)

Theorem: QBF is in PSPACE

Proof: Given $\varphi := Q_1 X_1 \ \ldots Q_n X_n \psi$, letting $m := |\psi|$

Eval-QBF(φ):

if $n = 0$ Accept if ψ evaluates to true. Reject otherwise.

if $\varphi := \exists X \psi'$
construct $\varphi_1 := \psi'[X \mapsto 1]$
if Eval-QBF(φ_1) evaluates to true, accept.
else construct $\varphi_0 := \psi'[X \mapsto 0]$ (reuse space in Eval-QBF(φ_1))
return Eval-QBF(φ_0)

if $\varphi := \forall X \psi'$
construct $\varphi_1 := \psi'[X \mapsto 1]$
if Eval-QBF(φ_1) evaluates to false, reject.
else construct $\varphi_0 := \psi'[X \mapsto 0]$ (reuse space in Eval-QBF(φ_1))
return Eval-QBF(φ_0)

Space complexity: Algorithm uses $\mathcal{O}(nm)$ tape cells.
(At depth d of recursion tree, remember d simplified versions of φ; can be improved to $\mathcal{O}(n + m)$ by remembering φ and d bits...)

Theorem: QBF is NPSPACE-hard

Let $\mathcal{L} \in \text{NPSPACE}$. We show $\mathcal{L} \leq_p \mathrm{QBF}$.

Let $\mathcal{M} := (Q, \Sigma, \Gamma, q_0, \Delta, F_a, F_r)$ be a TM deciding $\mathcal{L}$
such that $\mathcal{M}$ never uses more than $p(n)$ cells.

For each input $w \in \Sigma^*$, $|w| = n$, we construct a formula $\varphi_{\mathcal{M},w}$ such that

$\mathcal{M}$ accepts w if, and only if, $\varphi_{\mathcal{M},w}$ is true.

Theorem: QBF is NPSPACE-hard

Let $\mathcal{L} \in$ NPSPACE. We show $\mathcal{L} \leq_p \mathrm{QBF}$.

Let $\mathcal{M} := (Q, \Sigma, \Gamma, q_0, \Delta, F_a, F_r)$ be a TM deciding $\mathcal{L}$
such that $\mathcal{M}$ never uses more than $p(n)$ cells.

For each input $w \in \Sigma^*$, $|w| = n$, we construct a formula $\varphi_{\mathcal{M},w}$ such that

$\mathcal{M}$ accepts w if, and only if, $\varphi_{\mathcal{M},w}$ is true.

Describe configuration $(q, p, a_1 \dots a_{p(n)})$ by a set

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

of variables and the truth assignment β defined as

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

NPSPACE-Hardness of QBF

Consider the following formula $\textsc{Conf}(\mathcal{V})$ with free variables

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

$$\textsc{Conf}(\mathcal{V}) := \bigvee_{q \in Q} \Big(Q_q \wedge \bigwedge_{q' \neq q} \neg Q_{q'}\Big) \quad \wedge \quad \bigvee_{p \leq p(n)} \Big(P_p \wedge \bigwedge_{p' \neq p} \neg P_{p'}\Big) \wedge$$

$$\bigwedge_{1 \leq i \leq p(n)} \bigvee_{a \in \Gamma} \big(S_{a,i} \wedge \bigwedge_{b \neq a \in \Gamma} \neg S_{b,i}\big)$$

Definition. For any truth assignment β of $\mathcal{V}$ define config$(\mathcal{V}, \beta)$ as

$$\{(q, p, w_1 \dots w_{p(n)}) : \beta(Q_q) = \beta(P_p) = \beta(S_{w_i,i}) = 1, \forall i \leq p(n)\}$$

Lemma

If β satisfies $\textsc{Conf}(\mathcal{V})$ then $|config(\mathcal{V}, \beta)| = 1$.

NPSPACE-hardness of QBF

Definition. For an assignment β of $\mathcal{V}$ we defined $\text{config}(\mathcal{V}, \beta)$ as

$$\big\{(q, p, w_1 \dots w_{p(n)}) : \beta(Q_q) = \beta(P_p) = \beta(S_{w_i,i}) = 1, \forall i \leq p(n)\big\}$$

Lemma

If β satisfies $\textsc{Conf}(\mathcal{V})$ then $|\mathit{config}(\mathcal{V}, \beta)| = 1$.

Remark. β may be defined on other variables than those in $\mathcal{V}$.

$\text{config}(\mathcal{V}, \beta)$ is a potential configuration of $\mathcal{M}$, but it may not be reachable from the start configuration of $\mathcal{M}$ on input w.

Conversely: Every configuration $(q, p, w_1 \dots w_{p(n)})$ induces a satisfying assignment.

Consider the following formula $\textsc{Next}(\mathcal{V}, \mathcal{V}')$ defined as

$$\textsc{Conf}(\mathcal{V}) \wedge \textsc{Conf}(\mathcal{V}') \wedge \textsc{Nochange}(\mathcal{V}, \mathcal{V}') \wedge \textsc{Change}(\mathcal{V}, \mathcal{V}').$$

$$\textsc{Nochange} := \bigvee_{1 \leq p \leq p(n)} P_p \wedge \Big(\bigwedge_{\substack{i \neq p \\ a \in \Gamma}} (S_{a,i} \leftrightarrow S'_{a,i}) \Big)$$

$$\textsc{Change} := \bigvee_{1 \leq p \leq p(n)} \Big(P_p \wedge \bigvee_{\substack{q \in Q \\ a \in \Gamma}} \big(Q_q \wedge S_{a,p} \wedge \bigvee_{(q,a,q',b,m) \in \Delta} (Q'_{q'} \wedge S'_{b,p} \wedge P'_{\text{``}p+m\text{''}}) \big) \Big)$$

Lemma

For any assignment β defined on $\mathcal{V}, \mathcal{V}'$:

$$\beta \textit{ satisfies } \textsc{Next}(\mathcal{V}, \mathcal{V}') \iff \textit{config}(\mathcal{V}, \beta) \vdash_{\mathcal{M}} \textit{config}(\mathcal{V}', \beta)$$

NPSPACE-hardness of QBF

Define $\textsc{Path}_i(\mathcal{V}_1, \mathcal{V}_2)$:

$\mathcal{M}$ starting on config$(\mathcal{V}_1, \beta)$ can reach config$(\mathcal{V}_2, \beta)$ in $\leq 2^i$ steps.

For $i = 0$***:*** $\textsc{Path}_0 \quad := \quad \mathcal{V}_1 = \mathcal{V}_2 \quad \vee \quad \textsc{Next}(\mathcal{V}_1, \mathcal{V}_2)$

NPSPACE-hardness of QBF

Define $\textsc{Path}_i(\mathcal{V}_1, \mathcal{V}_2)$:

$\mathcal{M}$ starting on config$(\mathcal{V}_1, \beta)$ can reach config$(\mathcal{V}_2, \beta)$ in $\leq 2^i$ steps.

For $i = 0$***:*** $\textsc{Path}_0 \quad := \quad \mathcal{V}_1 = \mathcal{V}_2 \quad \vee \quad \textsc{Next}(\mathcal{V}_1, \mathcal{V}_2)$

For $i \to i+1$***:***

Idea: $\textsc{Path}_{i+1}(\mathcal{V}_1, \mathcal{V}_2) := \exists \mathcal{V} \Big[\textsc{Conf}(\mathcal{V}) \wedge \textsc{Path}_i(\mathcal{V}_1, \mathcal{V}) \wedge \textsc{Path}_i(\mathcal{V}, \mathcal{V}_2) \Big]$

NPSPACE-hardness of QBF

Define $\textsc{Path}_i(\mathcal{V}_1, \mathcal{V}_2)$:

$\mathcal{M}$ starting on config$(\mathcal{V}_1, \beta)$ can reach config$(\mathcal{V}_2, \beta)$ in $\leq 2^i$ steps.

For $i = 0$***:*** $\textsc{Path}_0 := \mathcal{V}_1 = \mathcal{V}_2 \lor \textsc{Next}(\mathcal{V}_1, \mathcal{V}_2)$

For $i \to i + 1$***:***

Idea: $\textsc{Path}_{i+1}(\mathcal{V}_1, \mathcal{V}_2) := \exists \mathcal{V} \Big[\textsc{Conf}(\mathcal{V}) \land \textsc{Path}_i(\mathcal{V}_1, \mathcal{V}) \land \textsc{Path}_i(\mathcal{V}, \mathcal{V}_2) \Big]$

Problem: $|\textsc{Path}_i| = \mathcal{O}(2^i)$ (Reduction would use exp. time/space)

NPSPACE-hardness of QBF

Define $\textsc{Path}_i(\mathcal{V}_1, \mathcal{V}_2)$:

$\mathcal{M}$ starting on config$(\mathcal{V}_1, \beta)$ can reach config$(\mathcal{V}_2, \beta)$ in $\leq 2^i$ steps.

For $i = 0$***:*** $\quad \textsc{Path}_0 \quad := \quad \mathcal{V}_1 = \mathcal{V}_2 \quad \vee \quad \textsc{Next}(\mathcal{V}_1, \mathcal{V}_2)$

For $i \rightarrow i + 1$***:***

Idea: $\textsc{Path}_{i+1}(\mathcal{V}_1, \mathcal{V}_2) := \exists \mathcal{V} \Big[\textsc{Conf}(\mathcal{V}) \wedge \textsc{Path}_i(\mathcal{V}_1, \mathcal{V}) \wedge \textsc{Path}_i(\mathcal{V}, \mathcal{V}_2) \Big]$

Problem: $|\textsc{Path}_i| = \mathcal{O}(2^i)$ (Reduction would use exp. time/space)

New Idea:

$$\textsc{Path}_{i+1}(\mathcal{V}_1, \mathcal{V}_2) := \quad \exists \mathcal{V}\ \textsc{Conf}(\mathcal{V}) \wedge$$

$$\forall \mathcal{Z}_1 \forall \mathcal{Z}_2 \Big(\Big(\begin{matrix} (\mathcal{Z}_1 = \mathcal{V}_1 \wedge \mathcal{Z}_2 = \mathcal{V}) \\ (\mathcal{Z}_1 = \mathcal{V} \wedge \mathcal{Z}_2 = \mathcal{V}_2) \end{matrix} \quad \vee \Big) \rightarrow \textsc{Path}_i(\mathcal{Z}_1, \mathcal{Z}_2) \Big)$$

NPSPACE-hardness of QBF

Define $\textsc{Path}_i(\mathcal{V}_1, \mathcal{V}_2)$:

$\mathcal{M}$ starting on config$(\mathcal{V}_1, \beta)$ can reach config$(\mathcal{V}_2, \beta)$ in $\leq 2^i$ steps.

For $i = 0$**:** $\quad \textsc{Path}_0 \;:=\; \mathcal{V}_1 = \mathcal{V}_2 \;\vee\; \textsc{Next}(\mathcal{V}_1, \mathcal{V}_2)$

For $i \to i+1$**:**

Idea: $\textsc{Path}_{i+1}(\mathcal{V}_1, \mathcal{V}_2) := \exists \mathcal{V} \Big[\textsc{Conf}(\mathcal{V}) \wedge \textsc{Path}_i(\mathcal{V}_1, \mathcal{V}) \wedge \textsc{Path}_i(\mathcal{V}, \mathcal{V}_2) \Big]$

Problem: $|\textsc{Path}_i| = \mathcal{O}(2^i)$ (Reduction would use exp. time/space)

New Idea:

$$\textsc{Path}_{i+1}(\mathcal{V}_1, \mathcal{V}_2) := \exists \mathcal{V}\; \textsc{Conf}(\mathcal{V}) \wedge \forall \mathcal{Z}_1 \forall \mathcal{Z}_2 \Big(\big(\begin{array}{l} (\mathcal{Z}_1 = \mathcal{V}_1 \wedge \mathcal{Z}_2 = \mathcal{V}) \\ (\mathcal{Z}_1 = \mathcal{V} \wedge \mathcal{Z}_2 = \mathcal{V}_2) \end{array} \vee \big) \to \textsc{Path}_i(\mathcal{Z}_1, \mathcal{Z}_2) \Big)$$

Lemma

For any assignment β defined on $\mathcal{V}_1, \mathcal{V}_2$: If β satisfies $\textsc{Path}_i(\mathcal{V}_1, \mathcal{V}_2)$, then config$(\mathcal{V}_2, \beta)$ is reachable from config$(\mathcal{V}_1, \beta)$ in $\leq 2^i$ steps.

NPSPACE-hardness of QBF

Path$_i(\mathcal{V}_1, \mathcal{V}_2)$**:**

$\mathcal{M}$ starting on config$(\mathcal{V}_1, \beta)$ can reach config$(\mathcal{V}_2, \beta)$ in $\leq 2^i$ steps.

Start and end configuration:

$$\textsc{Start}(\mathcal{V}) \quad := \quad \textsc{Conf}(\mathcal{V}) \wedge Q_{q_0} \ \wedge P_0 \ \wedge \bigwedge_{i=0}^{n-1} S_{w_i,i} \wedge \bigwedge_{i=n}^{p(n)} S_{\Box,i}$$

$$\textsc{End}(\mathcal{V}) \quad := \quad \textsc{Conf}(\mathcal{V}) \ \wedge \ \bigvee_{q \in F_a} Q_q$$

Lemma

Let C_{start} of $\mathcal{M}$ on input w.

1. *β satisfies $\textsc{Start}$ if, and only if, config$(\mathcal{V}, \beta) = C_{start}$*
2. *β satisfies $\textsc{End}$ if, and only if, config$(\mathcal{V}, \beta)$ is an accepting stop configuration.* *(may not be reachable from C_{start})*

NPSPACE-hardness of QBF

$\mathbf{Path}_i(\mathcal{V}_1, \mathcal{V}_2)$:

$\mathcal{M}$ starting on config$(\mathcal{V}_1, \beta)$ can reach config$(\mathcal{V}_2, \beta)$ in $\leq 2^i$ steps.

Start and end configuration:

$$\textsc{Start}(\mathcal{V}) \;:=\; \textsc{Conf}(\mathcal{V}) \wedge Q_{q_0} \wedge P_0 \wedge \bigwedge_{i=0}^{n-1} S_{w_i,i} \wedge \bigwedge_{i=n}^{p(n)} S_{\Box,i}$$

$$\textsc{End}(\mathcal{V}) \;:=\; \textsc{Conf}(\mathcal{V}) \wedge \bigvee_{q \in F_a} Q_q$$

Lemma

Let C_{start} of $\mathcal{M}$ on input w.

1. *β satisfies $\textsc{Start}$ if, and only if, config$(\mathcal{V}, \beta) = C_{start}$*
2. *β satisfies $\textsc{End}$ if, and only if, config$(\mathcal{V}, \beta)$ is an accepting stop configuration. (may not be reachable from C_{start})*

Putting it all together: $\mathcal{M}$ accepts w if, and only if,

$\varphi_{\mathcal{M},w} := \exists \mathcal{V}_1\ \exists \mathcal{V}_2\ \textsc{Start}(\mathcal{V}_1) \wedge \textsc{End}(\mathcal{V}_2) \wedge \textsc{Path}_{p(n)}(\mathcal{V}_1, \mathcal{V}_2)$ is true.

NPSPACE-hardness of QBF (to conclude)

Theorem

QBF is NPSPACE-hard.

Proof. Let $\mathcal{L} \in$ NPSPACE, we show $\mathcal{L} \leq_p \mathrm{QBF}$.

Let $\mathcal{M} := (Q, \Sigma, q_0, \Delta, F_a, F_r)$ be a TM deciding $\mathcal{L}$. $\mathcal{M}$ never uses more than $p(n)$ cells.

For each input $w \in \Sigma^*$, $|w| = n$, we construct (in poly time!) a formula $\varphi_{\mathcal{M},w}$ such that

$\mathcal{M}$ accepts w if, and only if, $\varphi_{\mathcal{M},w}$ is true.

Glossed over some detail: $\varphi_{\mathcal{M},w}$ is not in prenex form, can be manipulated into that. Also, quantifiers don't alternate $\forall/\exists/\forall/\exists \ldots$; that also can be fixed...

Alternation, Games

The Formula Game

Players: Played by two Players $\exists$ and $\forall$

Board: A formula φ in conjunctive normal form with variables $X_1, \ldots, X_n$

Moves: Players take turns in assigning truth values to $X_1, \ldots, X_n$ in order.

That is, player $\exists$ assigns values to "odd" variables $X_1, X_3, \ldots$

Winning condition: After all variables have been instantiated, $\exists$ wins if the formula evaluates to true. Otherwise $\forall$ wins.

The Formula Game

Players: Played by two Players $\exists$ and $\forall$

Board: A formula φ in conjunctive normal form with variables $X_1, \ldots, X_n$

Moves: Players take turns in assigning truth values to $X_1, \ldots, X_n$ in order.

That is, player $\exists$ assigns values to "odd" variables $X_1, X_3, \ldots$

Winning condition: After all variables have been instantiated, $\exists$ wins if the formula evaluates to true. Otherwise $\forall$ wins.

> **Formula Game**
>
> *Input:* A CNF formula φ in the variables $X_1, \ldots, X_n$
>
> *Problem:* Does $\exists$ have a winning strategy in the game on φ?

Theorem. FORMULA GAME is PSPACE-complete.

Geography

A generalised version of Geography:

The board is a directed graph G and a start node $s \in V(G)$

Initially the token is on the start node.

Players take turns in pushing this token along a directed edge.

If a player cannot move except to a node visited before, he loses.

Geography

A generalised version of Geography:

The board is a directed graph G and a start node $s \in V(G)$

Initially the token is on the start node.

Players take turns in pushing this token along a directed edge.

If a player cannot move except to a node visited before, he loses.

Geography

Input:	Directed graph G, start node $s \in V(G)$
Problem:	Does Player 1 have a winning strategy?

Theorem. GEOGRAPHY is PSPACE-complete.

(see blackboard or Sipser Theorem 8.14)

Alternating Turing Machines

Alternating Turing Machines

Definition. An alternating Turing machine $\mathcal{M}$ is a non-deterministic Turing accepter whose set of non-final states is partitioned into existential and universal states.

$Q_\exists$: set of existential states $\quad$ $Q_\forall$: set of universal states

Acceptance: Consider the computation tree $\mathcal{T}$ of $\mathcal{M}$ on w

Alternating Turing Machines

Definition. An alternating Turing machine $\mathcal{M}$ is a non-deterministic Turing accepter whose set of non-final states is partitioned into existential and universal states.

$Q_\exists$: set of existential states $\quad$ $Q_\forall$: set of universal states

Acceptance: Consider the computation tree $\mathcal{T}$ of $\mathcal{M}$ on w

A configuration C in $\mathcal{T}$ is eventually accepting if

- C is an accepting stop configuration, i.e. an accepting leaf of $\mathcal{T}$
- $C = (q, p, w)$ with $q \in Q_\exists$ and there is at least one eventually accepting successor configuration in $\mathcal{T}$
- $C = (q, p, w)$ with $q \in Q_\forall$ and all successor configurations of C in $\mathcal{T}$ are eventually accepting

$\mathcal{M}$ accepts w if the start configuration on w is eventually accepting.

Example: Alternating Algorithm for GEOGRAPHY

Input: Directed graph G $s \in V(G)$ start node.

Set $\textsc{Visited} := \{s\}$ Mark s as current node.

repeat

existential move: choose successor $v \notin \textsc{Visited}$ of current node s

if not possible **then** reject.

$\textsc{Visited} := \textsc{Visited} \cup \{v\}$

set current node $s := v$

universal move: choose successor $v \notin \textsc{Visited}$ of current node s

if not possible **then** accept.

$\textsc{Visited} := \textsc{Visited} \cup \{v\}$

set current node $s := v$

Note. This algorithm runs in alternating polynomial time.

Alternation as Model for Parallelism

Alternation can be seen as a form of parallelism:

universal move: choose successor $v \notin \textsc{Visited}$ of current node s

parallel computation:
in parallel, try for all successors $v \notin \textsc{Visited}$ of current node s

Universal moves are one possible way of modelling parallel computation.

Basic definitions of alternating time/space complexity

$\mathcal{L}(\mathcal{M})$ denotes words (in Σ^*) accepted by $\mathcal{M}$.

For function $T : \mathbb{N} \to \mathbb{N}$, an alternating TM is T time-bounded if every computation of $\mathcal{M}$ on input w of length n halts after $\leq T(n)$ steps.

Analogously for T space-bounded.

Basic definitions of alternating time/space complexity

$\mathcal{L}(\mathcal{M})$ denotes words (in Σ^*) accepted by $\mathcal{M}$.

For function $T : \mathbb{N} \to \mathbb{N}$, an alternating TM is T time-bounded if every computation of $\mathcal{M}$ on input w of length n halts after $\leq T(n)$ steps.

Analogously for T space-bounded.

For $T : \mathbb{N} \to \mathbb{N}$ a monotone growing function, define

1. ATIME(T) as the class of languages $\mathcal{L}$ for which there is a T-time bounded k-tape alternating Turing accepter deciding $\mathcal{L}$, $k \geq 1$.
2. ASPACE(T) as the class of languages $\mathcal{L}$ for which there is a T-space bounded alternating k-tape Turing accepter deciding $\mathcal{L}$, $k \geq 1$.

Alternating Complexity Classes:

Time classes:

- APTIME $:= \bigcup_{d\in\mathbb{N}}$ ATIME(n^d) — alternating poly time
- AEXPTIME $:= \bigcup_{d\in\mathbb{N}}$ ATIME(2^{n^d}) — alternating exp. time
- 2-AEXPTIME $:= \bigcup_{d\in\mathbb{N}}$ ATIME($2^{2^{n^d}}$)

Space classes:

- ALOGSPACE $:= \bigcup_{d\in\mathbb{N}}$ ASPACE($d \log n$)
- APSPACE $:= \bigcup_{d\in\mathbb{N}}$ ASPACE(n^d)
- AEXPSPACE $:= \bigcup_{d\in\mathbb{N}}$ ASPACE(2^{n^d})

Examples.

GEOGRAPHY $\in$ APTIME.

MONOTONE CVP (coming up next) $\in$ ALOGSPACE.

Similar alg.: CVP $\in$ ALOGSPACE.

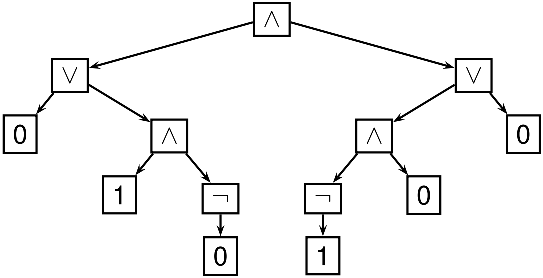
Example: Circuit Value Problem

Circuit. A connected directed acyclic graph with exactly one vertex of in-degree 0.

The vertices are labelled by:

label	no. of successors
$\wedge$	2
$\vee$	2
$\neg$	1
1	0
0	0

Example.



Evaluation of Circuits. A node v in a circuit C evaluates to 1 if

- v is a leaf labelled by 1
- v is a node labelled by $\vee$ and one successor evaluates to 1
- v is a node labelled by $\neg$ and its successor evaluates to 0
- v is a node labelled by $\wedge$ and both successors evaluate to 1

C evaluates to 1 if its root evaluates to 1.

Evaluation of Circuits. A node v in a circuit C evaluates to 1 if

- v is a leaf labelled by 1
- v is a node labelled by $\vee$ and one successor evaluates to 1
- v is a node labelled by $\neg$ and its successor evaluates to 0
- v is a node labelled by $\wedge$ and both successors evaluate to 1

C evaluates to 1 if its root evaluates to 1.

Circuit Value Problem.

CVP

Input:	Circuit C
Problem:	Does C evaluate to 1?

Monotone Circuit Value Problem.

Monotone CVP

Input:	Monotone circuit C without negation $\neg$.
Problem:	Does C evaluate to 1?

Monotone Circuit Value Problem

Input: Monotone circuit C with root s.

Set $\textsc{Current} := s$.

while $\textsc{Current}$ is not a leaf **do**

if current node v is a $\vee$-node **then**

existential move: choose successor v' of v

else if current node v is a $\wedge$-node **then**

universal move: choose successor v' of v

end if

set current node $\textsc{Current} := v'$

if $\textsc{Current}$ is labelled by 1 **then** accept **else** reject.

Monotone Circuit Value Problem

Input: Monotone circuit C with root s.

Set $\textsc{Current} := s$.

while $\textsc{Current}$ is not a leaf **do**

if current node v is a $\vee$-node **then**

existential move: choose successor v' of v

else if current node v is a $\wedge$-node **then**

universal move: choose successor v' of v

end if

set current node $\textsc{Current} := v'$

if $\textsc{Current}$ is labelled by 1 **then** accept **else** reject.

Note. This algorithm runs in alternating logarithmic space.

Basic general properties of alternating TMs/complexity

Non-determinism. A non-deterministic Turing accepter **is** an alternating TM (without universal states).

$$\mathcal{L} \in \mathsf{NP} \Longrightarrow \mathcal{L} \in \mathsf{APTIME}$$

Basic general properties of alternating TMs/complexity

Non-determinism. A non-deterministic Turing accepter **is** an alternating TM (without universal states).

$$\mathcal{L} \in \mathsf{NP} \implies \mathcal{L} \in \mathsf{APTIME}$$

Reductions. If $\mathcal{L} \in \mathsf{ATIME}(T)$ and $\mathcal{L}' \leq_p \mathcal{L}$ then $\mathcal{L}' \in \mathsf{ATIME}(T)$.

Hence: $\mathsf{PSPACE} \subseteq \mathsf{APTIME}$

(As GEOGRAPHY $\in$ APTIME.)

Basic general properties of alternating TMs/complexity

Non-determinism. A non-deterministic Turing accepter **is** an alternating TM (without universal states).

$$\mathcal{L} \in \mathsf{NP} \implies \mathcal{L} \in \mathsf{APTIME}$$

Reductions. If $\mathcal{L} \in \mathsf{ATIME}(T)$ and $\mathcal{L}' \leq_p \mathcal{L}$ then $\mathcal{L}' \in \mathsf{ATIME}(T)$.

Hence: $\mathsf{PSPACE} \subseteq \mathsf{APTIME}$

(As GEOGRAPHY $\in$ APTIME.)

Complementation. Alternating Turing accepters are easily "negated".

Let $\mathcal{M}$ be an alternating TM accepting language $\mathcal{L}$

Let $\mathcal{M}'$ be obtained from $\mathcal{M}$ by swapping

- the accepting and rejecting state
- swapping existential and universal states.

Then $\mathcal{L}(\mathcal{M}') = \overline{\mathcal{L}(\mathcal{M})}$

Example

Satisfiability for formulae $\varphi := \exists X_1 \forall X_2 \psi$, where ψ is quantifier-free:

Algorithm 1:

existential move. choose assignment $\beta : X_1 \mapsto 1$ or $\beta : X_1 \mapsto 0$.

universal move.

choose assignment $\beta := \beta \cup \{X_2 \mapsto 1\}$ or $\beta := \beta \cup \{X_2 \mapsto 0\}$.

if β satisfies ψ **then** accept **else** reject.

Example

Satisfiability for formulae $\varphi := \exists X_1 \forall X_2 \psi$, where ψ is quantifier-free:

Algorithm 1:

existential move. choose assignment $\beta : X_1 \mapsto 1$ or $\beta : X_1 \mapsto 0$.

universal move.

choose assignment $\beta := \beta \cup \{X_2 \mapsto 1\}$ or $\beta := \beta \cup \{X_2 \mapsto 0\}$.

if β satisfies ψ **then** accept **else** reject.

Its complement is defined as:

Algorithm 2:

universal move. choose assignment $\beta : X_1 \mapsto 1$ or $\beta : X_1 \mapsto 0$.

existential move.

choose assignment $\beta := \beta \cup \{X_2 \mapsto 1\}$ or $\beta := \beta \cup \{X_2 \mapsto 0\}$.

if β satisfies ψ **then** reject **else** accept.

Example

Satisfiability for formulae $\varphi := \exists X_1 \forall X_2 \psi$, where ψ is quantifier-free:

Algorithm 1:

existential move. choose assignment $\beta : X_1 \mapsto 1$ or $\beta : X_1 \mapsto 0$.

universal move.

choose assignment $\beta := \beta \cup \{X_2 \mapsto 1\}$ or $\beta := \beta \cup \{X_2 \mapsto 0\}$.

if β satisfies ψ **then** accept **else** reject.

Its complement is defined as:

Algorithm 2:

universal move. choose assignment $\beta : X_1 \mapsto 1$ or $\beta : X_1 \mapsto 0$.

existential move.

choose assignment $\beta := \beta \cup \{X_2 \mapsto 1\}$ or $\beta := \beta \cup \{X_2 \mapsto 0\}$.

if β satisfies ψ **then** reject **else** accept.

Note: Algorithm 1 accepts φ iff Algorithm 2 rejects φ

Alternating vs. Sequential Time and Space

Theorem

APTIME = PSPACE

Proof.

1. We have already seen that GEOGRAPHY ∈ APTIME.
   As GEOGRAPHY is PSPACE-complete,

$$\text{PSPACE} \subseteq \text{APTIME}.$$

Theorem

APTIME = PSPACE

Proof.

1. We have already seen that GEOGRAPHY ∈ APTIME.
   As GEOGRAPHY is PSPACE-complete,

   $$\text{PSPACE} \subseteq \text{APTIME}.$$

2. APTIME ⊆ PSPACE follows from the following more general result.

   Lemma. For $f(n) \geq n$ we have

   $$\text{ATIME}(f(n)) \subseteq \text{DSPACE}(f(n))$$

   (explore config. tree of ATM of depth $f(n)$)

Alternating vs. Sequential Time and Space

Theorem.

1. For $f(n) \geq n$ we have

$$\mathsf{ATIME}(f(n)) \subseteq \mathsf{DSPACE}(f(n)) \subseteq \mathsf{ATIME}(f^2(n))$$

2. For $f(n) \geq \log n$ we have $\mathsf{ASPACE}(f(n)) = \mathsf{DTIME}(2^{\mathcal{O}(f(n))})$

(see Sipser Thm. 10.21)

Lemma. For $f(n) \geq n$ we have $\mathsf{DSPACE}(f(n)) \subseteq \mathsf{ATIME}(f^2(n))$.

Proof. Let $\mathcal{L}$ be in DSPACE($f(n)$) and $\mathcal{M}$ be an $f(n)$ space-bounded TM deciding $\mathcal{L}$.

On input w, $\mathcal{M}$ makes at most $2^{\mathcal{O}(f(n))}$ computation steps.

Alternating Algorithm. $Reach(C_1, C_2, t)$
Returns 1 if C_2 is reachable from C_1 in $\leq 2^t$ steps.

$Reach(C_1, C_2, t)$
if $t = 0$ **do**
  if $C_1 = C_2$ or $C_1 \vdash C_2$ **do** return 1 **else** return 0 **od**
else
  existential step. choose configuration C with $|C| \leq \mathcal{O}(f(n))$
  universal step. choose $(D_1, D_2) = (C_1, C)$ or $(D_1, D_2) = (C, C_2)$
  return $Reach(D_1, D_2, t-1)$.
fi

Alternating vs. Sequential Time and Space

Theorem.

1. For $f(n) \geq n$ we have

$$\mathrm{ATIME}(f(n)) \subseteq \mathrm{DSPACE}(f(n)) \subseteq \mathrm{ATIME}(f^2(n))$$

2. For $f(n) \geq \log n$ we have $\mathrm{ASPACE}(f(n)) = \mathrm{DTIME}(2^{\mathcal{O}(f(n))})$

(see Sipser Thm. 10.21)

Alternating vs. Sequential Time and Space

Theorem.

1. For $f(n) \geq n$ we have

$$\mathrm{ATIME}(f(n)) \subseteq \mathrm{DSPACE}(f(n)) \subseteq \mathrm{ATIME}(f^2(n))$$

2. For $f(n) \geq \log n$ we have $\mathrm{ASPACE}(f(n)) = \mathrm{DTIME}(2^{\mathcal{O}(f(n))})$

(see Sipser Thm. 10.21)

Corollaries.

- ALOGSPACE = PTIME
- APTIME = PSPACE
- APSPACE = EXPTIME

The polynomial-time hierarchy

- NP: given an existentially-quantified QBF, is it true?
- co-NP: given a universally-quantified QBF, is it true?
- PSPACE: given an unrestricted QBF, is it true?

The polynomial-time hierarchy

- NP: given an existentially-quantified QBF, is it true?
- co-NP: given a universally-quantified QBF, is it true?
- PSPACE: given an unrestricted QBF, is it true?

Intermediate between NP/co-NP and PSPACE:

- Evaluate formula of the form $\exists x_1, \ldots, x_n \forall y_1, \ldots, y_n \ \varphi$
- Evaluate formula of the form $\forall x_1, \ldots, x_n \exists y_1, \ldots, y_n \ \varphi$
- Evaluate formula of the form $\exists x_1, \ldots, x_n \forall y_1, \ldots, y_n \exists z_1, \ldots, z_n \ \varphi$
- etc.

$\rightsquigarrow$ yet more complexity classes! (seemingly)

Sipser, chapter 10.3 (brief mention); Arora/Barak Chapter 5

Model of computation for (say) $\exists x_1, \ldots, x_n \forall y_1, \ldots, y_n \ \varphi$?

The polynomial-time hierarchy

Model of computation for (say) $\exists x_1, \ldots, x_n \forall y_1, \ldots, y_n\ \varphi$?

—Yes, poly-time alternating TM where $\exists$ states must precede $\forall$ states, in any computation.
Any such formula has ATM of this kind that solves it; any ATM can be converted to equivalent $\exists \ldots \forall$-formula.

—Another answer: in terms of oracle machines...

The polynomial-time hierarchy

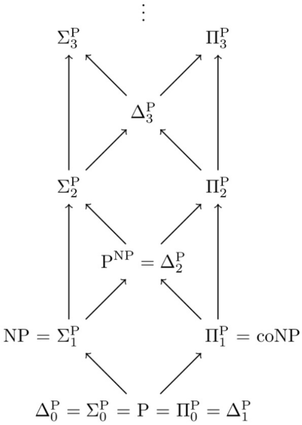


diagram taken from Wikipedia

$\Delta_{i+1}^{P} := P^{\Sigma_i^P}$
$\Sigma_{i+1}^{P} := NP^{\Sigma_i^P}$
$\Pi_{i+1}^{P} := \text{co-}NP^{\Sigma_i^P}$

A^B: problems solved by A-machine with oracle for B-complete problem

Warm-up: consider P^P, NP^P, P^{NP},...

The polynomial-time hierarchy

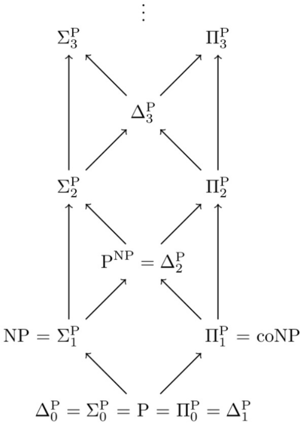


diagram taken from Wikipedia

$\Delta_{i+1}^{P} := P^{\Sigma_i^P}$
$\Sigma_{i+1}^{P} := NP^{\Sigma_i^P}$
$\Pi_{i+1}^{P} := \text{co-}NP^{\Sigma_i^P}$

A^B: problems solved by A-machine with oracle for B-complete problem

Warm-up: consider P^P, NP^P, P^{NP},...

P^{NP} seems to be more than just NP; indeed there are classes of interest intermediate between NP and P^{NP}!

The polynomial-time hierarchy

Some key facts:

- PH lies below PSPACE; if any problem is complete for PH, it must belong to the k-th level of the hierarchy, and PH would "collapse" to that level
- If P is equal to NP, then PH would collapse to P
- If NP is equal to co-NP, then PH collapses to that level. (hints that NP $\neq$ co-NP.)

(some proof details on board)