Computational Complexity; slides 3, HT 2023
Space complexity, time/space hierarchy theorems, PSPACE, alternating TMs

Paul W. Goldberg (Dept. of CS, Oxford)

HT 2023

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 $\subsetneq$ 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: Turing machines have a designated read only input tape. So, "logarithmic space" becomes meaningful.

Space Complexity

Definition. Let M be a Turing accepter and $S : \mathbb{N} \to \mathbb{N}$ a monotone growing function. M is <u>S-space bounded</u> if for all input words w, M halts and uses at most $S(|w|)$ non-input tape cells.

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

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

In the above defs, a single separate work-tape is sufficient.

Note:

$$
\begin{array}{ccccc}
\text{LOGSPACE} & \subseteq & \text{PSPACE} & \subseteq & \text{EXPSPACE} \\
\rotatebox{90}{$\supseteq$} & & \rotatebox{90}{$\supseteq$} & & \rotatebox{90}{$\supseteq$} \\
\text{NLOGSPACE} & \subseteq & \text{NPSPACE} & \subseteq & \text{NEXPSPACE}
\end{array}
$$

Elementary relationships between time and space

Easy observation:

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

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

A bit harder:

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

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

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 $f : \mathbb{N} \to \mathbb{N}$:

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

Proof. Based on *configuration graphs* (next 2 slides) 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 $M := (Q, \Sigma, \Gamma, q_0, \Delta, F_a, F_r)$ be a 1-tape Turing accepter.
(plus input tape)

Recall: Configuration of 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 M, $n := |w|$

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

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

Vertices:	All possible configurations of M up to length $f(\|w\|)$
Edges:	Edge $(C_1, C_2) \in E(\mathcal{G}(M, w))$, if $C_1 \vdash_M C_2$

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

Hence, to test if M accepts input w,

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

Recall: L denotes LOGSPACE; NL=NLOGSPACE

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

Simulating non-deterministic computations with limited space

Notice that SAT can be solved in *linear* space

Just try every possible assignment, reusing space.

Simulating non-deterministic computations with limited space

Notice that SAT can be solved in *linear* space

Just try every possible assignment, reusing space.

Hence 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.

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

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

Time-constructible (also called "proper") complexity function f:
$x \mapsto f(|x|)$ can be computed in time $O(f(n))$; $f(n) \geq n$; $f(n+1) \geq f(n)$

Theorem

If f, g are time-constructible, and $f(n) \log f(n) = o(g(n))$, then $\mathrm{TIME}(f(n)) \subsetneq \mathrm{TIME}(g(n))$.

Let's prove something weaker:

For time-constructible $f(n) \geq n$, we have

$$\mathrm{TIME}(f(n)) \subsetneq \mathrm{TIME}((f(2n+1))^3).$$

It follows that P is a proper subset of EXPTIME.

Proof sketch: consider "time-bounded halting language"

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

H_f belongs to $\mathrm{TIME}((f(n))^3)$: construct a universal TM that uses "quadratic overhead" to simulate a step of $\mathcal{M}$.

Time Hierarchy theorem

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

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

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:
  $D_f(M)$: 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.

Theorem. (Space Hierarchy Theorem)

Let $S, s : \mathbb{N} \to \mathbb{N}$ be functions such that

1. S is space constructible, and
2. $S(n) \geq n$,
3. $s = o(S)$.

Then $\mathrm{DSPACE}(s) \subsetneq \mathrm{DSPACE}(S)$.

Reminder: item 3 means that $\lim_{n\to\infty}(s(n)/S(n)) = 0$.

Proof later, but note consequences: LOGSPACE is a proper subset of PSPACE, is proper subset of EXPSPACE

Usage of a "padding" technique

Theorem

If P=NP, *then* EXPTIME=NEXPTIME

Suppose X ∈NEXPTIME. Define $pad(X)$ as follows:

$$w \in X \text{ iff } w\square^{2^n} \in pad(X) \quad (\text{where } n = |w|)$$

We have $pad(X)$ ∈NP: Given a word of the form $w\square^N$,

- Check you have the right number of $\square$'s.
- run the NEXPTIME algorithm on w-prefix (not the $\square$'s).

Hence $pad(X)$ ∈P by assumption.
Then, you can take poly-time algorithm for $pad(X)$, and convert it to algorithm that checks w-prefix, in time exponential in $|w|$.

Savitch's Theorem: PSPACE=NPSPACE

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

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

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

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

Savitch's Theorem

More generally:

Theorem Savitch 1970

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

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

It follows that PSPACE = NPSPACE; EXPSPACE = NEXPSPACE

A PSPACE-complete problem: QBF

c.f. Cook's theorem.
A more general kind of logic problem characterises PSPACE
`https://en.wikipedia.org/wiki/True_quantified_Boolean_formula`

A Quantified Boolean Formula is a formula of the form

$$Q_1 X_1 \dots Q_n X_n \varphi(X_1, \dots, X_n)$$

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

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

Consider the following problem:

QBF	
Input:	A QBF formula φ.
Question:	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.

Similarly, QBF is also co-NP-hard.

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)

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

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

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

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

Theorem: QBF is NPSPACE-hard

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

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

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

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

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

$$\mathcal{V} := \{X_q, Y_i, Z_{a,i} : q \in Q, \quad a \in \Gamma, \quad 0 \leq i < p(n)\}$$

of variables and the truth assignment β defined as

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

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

$$\mathcal{V} := \{X_q, Y_i, Z_{a,i} : q \in Q, \quad a \in \Gamma, \quad 0 \leq i < p(n)\}$$

$$\textsc{Conf}(\mathcal{V}) := \bigvee_{q \in Q} \Big(X_q \wedge \bigwedge_{q' \neq q} \neg X_{q'} \Big) \quad \wedge \quad \bigvee_{p \leq p(n)} \Big(Y_p \wedge \bigwedge_{p' \neq p} \neg Y_{p'} \Big) \wedge$$

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

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

$$\{(q, p, w_1 \dots w_{p(n)}) : \beta(X_q) = \beta(Y_p) = \beta(Z_{w_i,i}) = 1, \forall i \leq p(n)\}$$

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 config$(\mathcal{V}, \beta)$ as

$$\{(q, p, w_1 \dots w_{p(n)}) : \beta(X_q) = \beta(Y_p) = \beta(Z_{w_i,i}) = 1, \forall i \leq p(n)\}$$

Lemma

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

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

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

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

NPSPACE-Hardness of QBF

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

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

$$\textsc{NoChange} := \bigwedge_{1 \leq p \leq p(n)} \Big(Y_p \Rightarrow \bigwedge_{\substack{i \neq p \\ a \in \Gamma}} (Z_{a,i} \leftrightarrow Z'_{a,i}) \Big)$$

$$\textsc{Change} := \bigwedge_{1 \leq p \leq p(n)} \Big((Y_p \wedge X_q \wedge Z_{a,p}) \Rightarrow$$

$$\bigvee_{(q,a,q',b,m) \in \Delta} (X'_{q'} \wedge Z'_{b,p} \wedge Y'_{\text{“}p+m\text{”}}) \Big)$$

Lemma

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

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

NPSPACE-hardness of QBF

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

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

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

NPSPACE-hardness of QBF

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

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

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

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

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

NPSPACE-hardness of QBF

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

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

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:

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

NPSPACE-hardness of QBF

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

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

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:

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

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

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

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

Start and end configuration:

$$\textsc{Start}(\mathcal{V}) \quad := \quad \textsc{Conf}(\mathcal{V}) \wedge X_{q_0} \;\wedge Y_0 \;\wedge \bigwedge_{i=0}^{n-1} Z_{w_i,i} \wedge \bigwedge_{i=n}^{p(n)} Z_{\Box,i}$$

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

Lemma

Let C_{start} be starting configuration of 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. (not nec reachable from C_{start})*

NPSPACE-hardness of QBF

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

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

Start and end configuration:

$$\textsc{Start}(\mathcal{V}) \quad := \quad \textsc{Conf}(\mathcal{V}) \wedge X_{q_0} \wedge Y_0 \wedge \bigwedge_{i=0}^{n-1} Z_{w_i,i} \wedge \bigwedge_{i=n}^{p(n)} Z_{\square,i}$$

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

Lemma

Let C_{start} be starting configuration of 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. (not nec reachable from C_{start})*

Putting it all together: M accepts w if, and only if,

$\varphi_{M,w} := \exists \mathcal{V}_1\ \exists \mathcal{V}_2\ \textsc{Start}(\mathcal{V}_1) \wedge \textsc{End}(\mathcal{V}_2) \wedge \textsc{Path}_{p(n)}(\mathcal{V}_1, \mathcal{V}_2)$ 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 $M := (Q, \Sigma, q_0, \Delta, F_a, F_r)$ be a TM deciding $\mathcal{L}$. M never uses more than $p(n)$ cells.

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

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

Glossed over some detail: $\varphi_{M,w}$ 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...

To conclude

We have a "natural" PSPACE-complete problem

"natural" (slightly vague definition): the problem does not arise in the study of PSPACE, it has separate interest.

obvious analogy with SAT being complete for NP

Next: how to use this to prove various other problems are also PSPACE-complete.
Example of PSPACE-completeness (the "geography" game)
Then, alternative characterisation of PSPACE (as poly-time "alternating" TM). Recall general point that when there are various characterisations of a complexity class, it suggests the class is important.
Afterwards, polynomial hierarchy (classes between NP/co-NP and PSPACE)

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.

Formula Game (extended version)

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

- After players have chosen values for the variables, player $\forall$ chooses a clause
- Then player $\exists$ chooses a literal within that clause
- *exists* wins if the literal is satisfied, else $\forall$ wins

Example

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

if $\exists$-player makes right choices, for all clauses C, there exists, within C, a satisfied literal

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.

Edges may not be used more than once. If a player cannot move, 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.

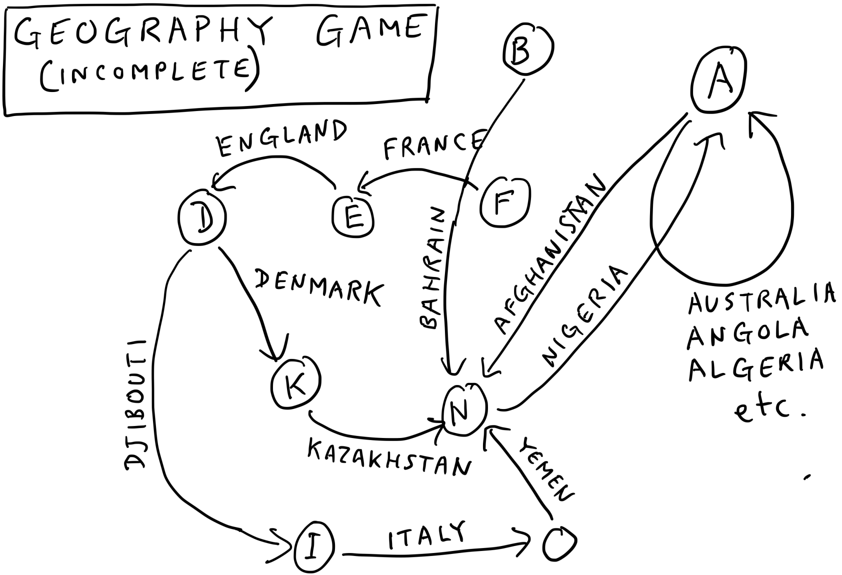
Players take turns in pushing this token along a directed edge.

Edges may not be used more than once. If a player cannot move, 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.

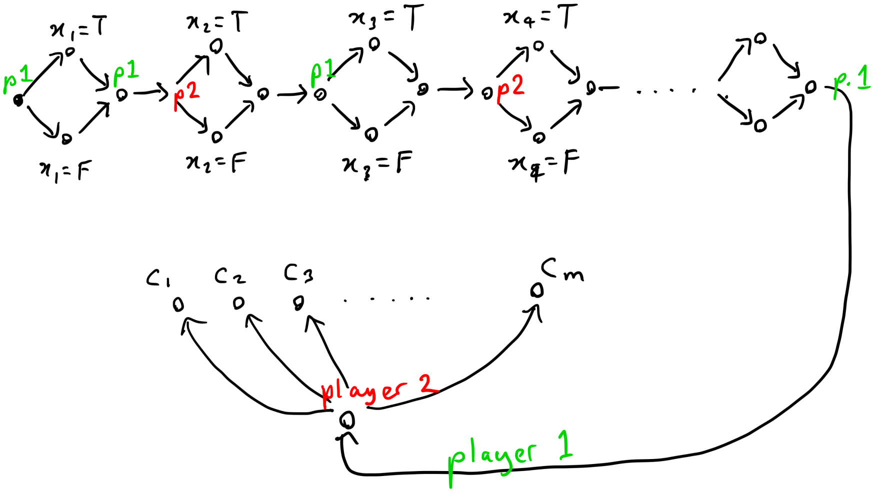
(Sipser Theorem 8.14)


GEOGRAPHY GAME
(INCOMPLETE)
B
A
ENGLAND
FRANCE
D
E
F
DENMARK
BAHRAIN
AFGHANISTAN
NIGERIA
AUSTRALIA
ANGOLA
ALGERIA
etc.
DJIBOUTI
K
N
KAZAKHSTAN
YEMEN
I
ITALY
O

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots \phi(x)$$

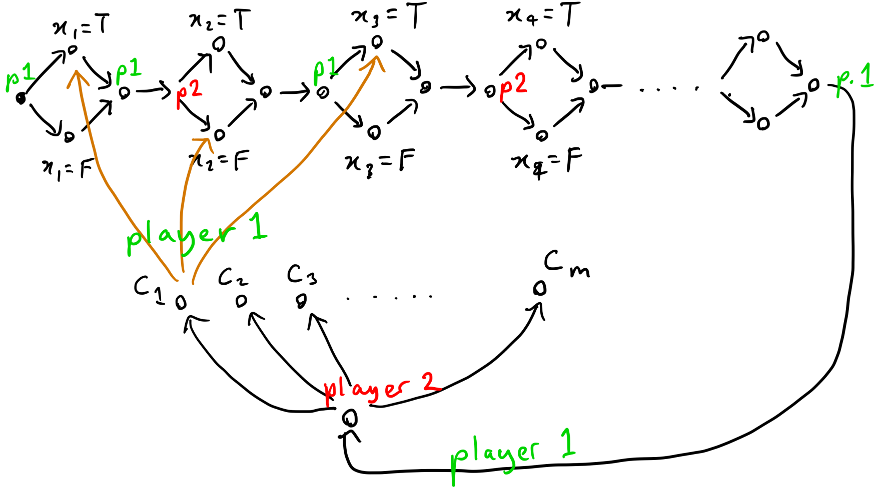
$$\phi = C_1 \wedge C_1 \wedge \ldots \wedge C_m$$

player 1 wants T
player 2 wants F
winner plays last



$\boxed{\text{Suppose } C_1 = x_1 \vee \neg x_2 \vee x_3}$

player 1 wants T
player 2 wants F
winner plays last



Next: Alternating Turing Machines

general idea: a class of automata whose languages are all the PSPACE languages. They can be a useful way to prove membership of problems in PSPACE.

They also give alternative characterisations of P, EXPTIME

Definition. An alternating Turing machine 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 M on w

Definition. An alternating Turing machine 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 M on w

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

- C is an accepting stop configuration: 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

M accepts w if 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.

Basic definitions of alternating time/space complexity

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

For function $T : \mathbb{N} \to \mathbb{N}$, an alternating TM is T time-bounded if every computation of 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

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

For function $T : \mathbb{N} \to \mathbb{N}$, an alternating TM is T time-bounded if every computation of 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 increasing 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 $:= \bigcup_{d \in \mathbb{N}} \mathrm{ATIME}(n^d)$ alternating poly time
- AEXPTIME $:= \bigcup_{d \in \mathbb{N}} \mathrm{ATIME}(2^{n^d})$ alternating exp. time
- 2-AEXPTIME $:= \bigcup_{d \in \mathbb{N}} \mathrm{ATIME}(2^{2^{n^d}})$

Space classes:

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

Examples.

GEOGRAPHY $\in$ APTIME.

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

Similar alg.: CVP $\in$ ALOGSPACE.

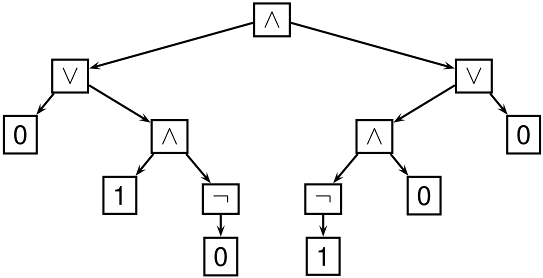
Circuit Value Problem

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 Current $:= s$.

while 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 Current := v'

if Current is labelled by 1 **then** accept **else** reject.

Monotone Circuit Value Problem

Input: Monotone circuit C with root s.

Set Current $:= s$.

while 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 Current $:=$ v'

if Current is labelled by 1 **then** accept **else** reject.

Note. This algorithm runs in alternating logarithmic space. Can be extended to general CVP

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).

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

Reductions. If $\mathcal{L} \in \mathsf{ATIME}(T)$ and $\mathcal{L}' \leq_p \mathcal{L}$ then $\mathcal{L}' \in \mathsf{ATIME}(T + f)$ where f is a polynomial.

Since GEOGRAPHY is PSPACE-complete and also in APTIME we have $\mathsf{PSPACE} \subseteq \mathsf{APTIME}$

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 $\mathcal{L}' \in \mathsf{ATIME}(T+f)$ where f is a polynomial.

Since GEOGRAPHY is PSPACE-complete and also in APTIME we have $\mathsf{PSPACE} \subseteq \mathsf{APTIME}$

Complementation. Alternating Turing accepters are easily "negated".

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

Let M' be obtained from M by swapping

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

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

Example of complementation

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 $\beta := \beta \cup \{X_2 \mapsto 1\}$ and $\beta := \beta \cup \{X_2 \mapsto 0\}$.

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

Example of complementation

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 $\beta := \beta \cup \{X_2 \mapsto 1\}$ and $\beta := \beta \cup \{X_2 \mapsto 0\}$.

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 of complementation

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 $\beta := \beta \cup \{X_2 \mapsto 1\}$ and $\beta := \beta \cup \{X_2 \mapsto 0\}$.

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

Theorem

APTIME = PSPACE

Proof.

1. We have already seen that GEOGRAPHY $\in$ APTIME.
   As GEOGRAPHY is PSPACE-complete,

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

Alternating vs. Sequential Time and Space

Theorem

APTIME = PSPACE

Proof.

1. We have already seen that GEOGRAPHY $\in$ APTIME.
   As GEOGRAPHY is PSPACE-complete,

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

2. APTIME $\subseteq$ PSPACE follows from the following more general result.

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

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

   To prove this, explore configuration tree of ATM of depth $f(n)$

Alternating vs. Sequential Time and Space

Theorem (more general)

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

Next slide: prove the containment shown in red

(c.f. Savitch's theorem)

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

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

On input w, 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.

if $t = 0$
  if $C_1 = C_2$ or $C_1 \vdash C_2$ **do** return 1 **else** return 0
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*$(D_1, D_2, t-1)$.

Alternating vs. Sequential Time and Space

Theorem (more general)

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)

Corollaries.

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

To conclude

Alternating TMs give us a different characterisation of complexity classes we have seen.

Next: the polynomial hierarchy: a sequence of classes that are intermediate between NP and PSPACE. They represent some important problems that are "above" NP and "below" PSPACE