Computational complexity

lecture 9

Polynomial hierarchy

The following problem is in **NP**:

INDSET = {(G,k) : in graph G there is an independent set of size $\geq k$}

Consider now a slightly more difficult problem:

EXACT-INDSET = {(G,k) : the largest independent set in G is of size k}

We see no reason for this problem to be in **NP**...

What would be a witness?

Polynomial hierarchy

EXACT-INDSET = $\{(G,k)$: the largest independent set in G is of size $k\}$

A similar problem:

MIN−DNF = { ϕ : ϕ is a formula in the DNF form, not equivalent to any smaller formula in the DNF form}

$= \{ \phi : \forall\, \psi,\ |\psi| < |\phi| \Rightarrow \exists$ valuation s such that $\phi(s) \neq \psi(s)\}$

In order to describe these problems, it is not enough to use one „exists” quantifier (as in **NP**), neither one „for all” quantifier (as in **coNP**). We have here a combination of two quantifiers.

Polynomial hierarchy

EXACT-INDSET = {(G,k) : the largest independent set in G is of size k}

A similar problem:

MIN−DNF = { ϕ : ϕ is a formula in the DNF form, not equivalent to any smaller formula in the DNF form}

= { ϕ : ∀ ψ, |ψ| < |ϕ| ⇒ ∃ valuation s such that ϕ(s)≠ψ(s)}

In order to describe these problems, it is not enough to use one „exists” quantifier (as in **NP**), neither one „for all” quantifier (as in **coNP**). We have here a combination of two quantifiers.

Class $\mathbf{\Sigma}_2^p$ contains languages L for which there is a machine M working in polynomial time, and a polynomial q such that:

$$x \in L \Leftrightarrow \exists\ u \in \{0,1\}^{q(|x|)}\ \forall\ v \in \{0,1\}^{q(|x|)}\ M(x,u,v)=1$$

The language EXACT-INDSET is of this form:

$\exists S\ \forall S'$. S is an independent set of size k and S' is not an independent set of size $>k$

Polynomial hierarchy

Class $\mathbf{\Sigma}_2^p$ contains languages L for which there is a machine M working in polynomial time, and a polynomial q such that:

$$x \in L \Leftrightarrow \exists\, u \in \{0,1\}^{q(|x|)}\ \forall\, v \in \{0,1\}^{q(|x|)}\ M(x,u,v)=1$$

The language EXACT-INDSET is of this form

Class $\mathbf{\Pi}_2^p$ contains complements of languages from $\mathbf{\Sigma}_2^p$; it is easy to see that it contains languages L for which there is a machine M working in polynomial time, and a polynomial q such that:

$$x \in L \Leftrightarrow \forall\, u \in \{0,1\}^{q(|x|)}\ \exists\, v \in \{0,1\}^{q(|x|)}\ M(x,u,v)=1$$

The language EXACT-INDSET is of this form as well:

$\forall S'\ \exists S$. S is an independent set of size k and

S' is not an independent set of size $>k$

Also the language MIN−DNF is of this form:

$\forall\ \psi\ \exists s\ .\ |\psi| < |\phi| \Rightarrow \phi(s) \neq \psi(s)$

However, it is believed that MIN-DNF does not belong to $\mathbf{\Sigma}_2^p$

Polynomial hierarchy

Class $\mathbf{\Sigma}^{p}_{k}$ contains languages L for which there is a machine M working in polynomial time, and a polynomial q such that:

$$x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{q(|x|)}\ \forall u_2 \in \{0,1\}^{q(|x|)} \ldots Q u_k \in \{0,1\}^{q(|x|)}\ .\ M(x,u_1,\ldots,u_k)=1$$

Class $\mathbf{\Pi}^{p}_{k}$ contains complements of languages from $\mathbf{\Sigma}^{p}_{k}$, i.e., languages L for which there is a machine M working in polynomial time, and a polynomial q such that:

$$x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)}\ \exists u_2 \in \{0,1\}^{q(|x|)} \ldots Q u_k \in \{0,1\}^{q(|x|)}\ .\ M(x,u_1,\ldots,u_k)=1$$

We also define **PH**$=\cup_k \mathbf{\Sigma}^{p}_{k}$

Polynomial hierarchy

Class $\mathbf{\Sigma}^p_k$ contains languages L for which there is a machine M working in polynomial time, and a polynomial q such that:

$$x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{q(|x|)}\ \forall u_2 \in \{0,1\}^{q(|x|)} \ldots Q u_k \in \{0,1\}^{q(|x|)}.\ M(x,u_1,\ldots,u_k)=1$$

Class $\mathbf{\Pi}^p_k$ contains complements of languages from $\mathbf{\Sigma}^p_k$, i.e., languages L for which there is a machine M working in polynomial time, and a polynomial q such that:

$$x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)}\ \exists u_2 \in \{0,1\}^{q(|x|)} \ldots Q u_k \in \{0,1\}^{q(|x|)}.\ M(x,u_1,\ldots,u_k)=1$$

We also define **PH**$=\cup_k \mathbf{\Sigma}^p_k$

Fact 1

Class $\mathbf{\Sigma}^p_k$ contains precisely languages recognizable in polynomial time by nondeterministic Turing machines with an oracle for a problem from $\mathbf{\Sigma}^p_{k-1}$, and $\mathbf{\Pi}^p_k$ contains their complements.

Polynomial hierarchy

Fact 1

Class $\mathbf{\Sigma}^p_k$ contains precisely languages recognizable in polynomial time by nondeterministic Turing machines with an oracle for a problem from $\mathbf{\Sigma}^p_{k-1}$, and $\mathbf{\Pi}^p_k$ contains their complements.

Proof

Let $L \in \mathbf{\Sigma}^p_k$. By definition there is a machine M working in polynomial time, and a polynomial q such that:

$$x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{q(|x|)}\ \forall u_2 \in \{0,1\}^{q(|x|)} \ldots Q u_k \in \{0,1\}^{q(|x|)} .\ M(x,u_1,\ldots,u_k)=1$$

Consider the language L' defined by

$$(x,u_1) \in L' \Leftrightarrow \forall u_2 \in \{0,1\}^{q(|x|)} \ldots Q u_k \in \{0,1\}^{q(|x|)} .\ M(x,u_1,\ldots,u_k)=1$$

The complement of L' is in $\mathbf{\Sigma}^p_{k-1}$.

It is easy to recognize L by a nondeterministic machine with oracle for (the complement of) L'.

Polynomial hierarchy

Fact 1

Class Σ^p_k contains precisely languages recognizable in polynomial time by nondeterministic Turing machines with an oracle for a problem from Σ^p_{k-1}, and Π^p_k contains their complements.

Proof

Let L be recognized by a nondet. machine N with oracle for $L'\in\Sigma^p_{k-1}$.

By definition there is a machine M' working in polynomial time, and a polynomial q' such that:

$$y\in L' \Leftrightarrow \exists v_1\in\{0,1\}^{q'(|y|)}\ \forall v_2\in\{0,1\}^{q'(|y|)} \ldots \overline{Q} v_{k-1}\in\{0,1\}^{q'(|y|)}.\ M'(y,v_1,\ldots,v_{k-1})=1$$

We observe that (for an appropriate polynomial q)

$$x\in L \Leftrightarrow \exists u_1\in\{0,1\}^{q(|x|)}\ \forall u_2\in\{0,1\}^{q(|x|)} \ldots Q u_k\in\{0,1\}^{q(|x|)}.\ M(x,u_1,\ldots,u_k)=1$$

where M checks that:

- a prefix of u_1 is of the form R, $v_{1,1}, \ldots, v_{1,n}$, where R is a run of N
- if y is the i-th query to L' in R with answer yes, $M'(y,v_{1,1},u_2,\ldots,u_{k-1})=1$
- if y is a query to L' in R with answer no, $M'(y,u'_2,\ldots,u'_k)=0$ (where $u'_2,\ldots,u'_k$ are prefixes of $u_2,\ldots,u_k$ of length $q'(y)$)

Thus $L\in\Sigma^p_k$.

Polynomial hierarchy

Fact 1

Class Σ^p_k contains precisely languages recognizable in polynomial time by nondeterministic Turing machines with an oracle for a problem from Σ^p_{k-1}, and Π^p_k contains their complements.

In particular:

- Σ^p_1=**NP**
- Π^p_1=**coNP**
- Σ^p_2 is sometimes denoted $\mathbf{NP}^{\mathbf{NP}}$ (**NP** with oracle in **NP**)
- Σ^p_2 contains in particular all languages from **NP** and from **coNP**

Polynomial hierarchy

Class $\mathbf{\Sigma}^{p}_{k}$ contains languages L for which there is a machine M working in polynomial time, and a polynomial q such that:

$$x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{q(|x|)}\ \forall u_2 \in \{0,1\}^{q(|x|)} \ldots Q u_k \in \{0,1\}^{q(|x|)} .\ M(x,u_1,\ldots,u_k)=1$$

Class $\mathbf{\Pi}^{p}_{k}$ contains complements of languages from $\mathbf{\Sigma}^{p}_{k}$, i.e., languages L for which there is a machine M working in polynomial time, and a polynomial q such that:

$$x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)}\ \exists u_2 \in \{0,1\}^{q(|x|)} \ldots Q u_k \in \{0,1\}^{q(|x|)} .\ M(x,u_1,\ldots,u_k)=1$$

We also define **PH**$=\cup_k \mathbf{\Sigma}^{p}_{k}$

How are these classes related?

Polynomial hierarchy

Class $\mathbf{\Sigma}_k^p$ contains languages L for which there is a machine M working in polynomial time, and a polynomial q such that:

$$x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{q(|x|)}\ \forall u_2 \in \{0,1\}^{q(|x|)} \dots Q u_k \in \{0,1\}^{q(|x|)} .\ M(x,u_1,\dots,u_k)=1$$

Class $\mathbf{\Pi}_k^p$ contains complements of languages from $\mathbf{\Sigma}_k^p$, i.e., languages L for which there is a machine M working in polynomial time, and a polynomial q such that:

$$x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)}\ \exists u_2 \in \{0,1\}^{q(|x|)} \dots Q u_k \in \{0,1\}^{q(|x|)} .\ M(x,u_1,\dots,u_k)=1$$

We also define **PH**$=\cup_k \mathbf{\Sigma}_k^p$

How are these classes related?

Fact 2: $\mathbf{\Sigma}_k^p \subseteq \mathbf{\Sigma}_{k+1}^p$, $\mathbf{\Sigma}_k^p \subseteq \mathbf{\Pi}_{k+1}^p$, $\mathbf{\Pi}_k^p \subseteq \mathbf{\Sigma}_{k+1}^p$, $\mathbf{\Pi}_k^p \subseteq \mathbf{\Pi}_{k+1}^p$

Proof: Obvious (follows from Fact 1)

Polynomial hierarchy

Fact 2: $\mathbf{\Sigma}^p_k \subseteq \mathbf{\Sigma}^p_{k+1}$, $\mathbf{\Sigma}^p_k \subseteq \mathbf{\Pi}^p_{k+1}$, $\mathbf{\Pi}^p_k \subseteq \mathbf{\Sigma}^p_{k+1}$, $\mathbf{\Pi}^p_k \subseteq \mathbf{\Pi}^p_{k+1}$

$$
\begin{tikzcd}[column sep=small]
 & \mathbf{NP}=\mathbf{\Sigma}^p_1 \arrow[r, phantom, "\subseteq"] \arrow[dr, phantom, sloped, "\subseteq"] & \mathbf{\Sigma}^p_2 \arrow[r, phantom, "\subseteq"] \arrow[dr, phantom, sloped, "\subseteq"] & \mathbf{\Sigma}^p_3 \arrow[r, phantom, "\subseteq"] & \cdots \\
\mathbf{P} \arrow[ur, phantom, sloped, "\subseteq"] \arrow[dr, phantom, sloped, "\subseteq"] & & & & \\
 & \mathbf{coNP}=\mathbf{\Pi}^p_1 \arrow[r, phantom, "\subseteq"] \arrow[ur, phantom, sloped, "\subseteq"] & \mathbf{\Pi}^p_2 \arrow[r, phantom, "\subseteq"] \arrow[ur, phantom, sloped, "\subseteq"] & \mathbf{\Pi}^p_3 \arrow[r, phantom, "\subseteq"] & \cdots
\end{tikzcd}
$$

Are these inclusions strict? And how are $\mathbf{\Sigma}^p_k$ and $\mathbf{\Pi}^p_k$ related?

Polynomial hierarchy

Fact 2: $\Sigma_k^p \subseteq \Sigma_{k+1}^p$, $\Sigma_k^p \subseteq \Pi_{k+1}^p$, $\Pi_k^p \subseteq \Sigma_{k+1}^p$, $\Pi_k^p \subseteq \Pi_{k+1}^p$

$$\begin{tikzcd}[column sep=small]
 & \mathbf{NP}=\Sigma_1^p \arrow[r, phantom, "\subseteq"] \arrow[ddr, phantom, "\subseteq" sloped] & \Sigma_2^p \arrow[r, phantom, "\subseteq"] \arrow[ddr, phantom, "\subseteq" sloped] & \Sigma_3^p \arrow[r, phantom, "\subseteq"] \arrow[ddr, phantom, "\subseteq" sloped] & \cdots \\
\mathbf{P} \arrow[ur, phantom, "\subseteq" sloped] \arrow[dr, phantom, "\subseteq" sloped] & & & & \\
 & \mathbf{coNP}=\Pi_1^p \arrow[r, phantom, "\subseteq"] \arrow[uur, phantom, "\subseteq" sloped] & \Pi_2^p \arrow[r, phantom, "\subseteq"] \arrow[uur, phantom, "\subseteq" sloped] & \Pi_3^p \arrow[r, phantom, "\subseteq"] \arrow[uur, phantom, "\subseteq" sloped] & \cdots
\end{tikzcd}$$

Are these inclusions strict? And how are Σ_k^p and Π_k^p related?

We don't know (it is believed that all these classes are different).

But there are only two possibilities:

- either all the classes are different, or
- they are different to some point, and then they start to be equal

Fact 3:

If $\Sigma_k^p = \Pi_k^p$, then $\Sigma_k^p = \Sigma_{k+1}^p = \ldots = \Pi_k^p = \Pi_{k+1}^p = \ldots = \mathbf{PH}$.

If **P**=**NP**, then $\mathbf{P} = \Sigma_1^p = \Sigma_2^p = \ldots = \Pi_1^p = \Pi_2^p = \ldots = \mathbf{PH}$.

Polynomial hierarchy

Fact 3:

If $\mathbf{\Sigma}^p_k=\mathbf{\Pi}^p_k$, then $\mathbf{\Sigma}^p_k=\mathbf{\Sigma}^p_{k+1}=...=\mathbf{\Pi}^p_k=\mathbf{\Pi}^p_{k+1}=...=$ **PH**.

If **P**=**NP**, then **P**$=\mathbf{\Sigma}^p_1=\mathbf{\Sigma}^p_2=...=\mathbf{\Pi}^p_1=\mathbf{\Pi}^p_2=...=$ **PH**.

Proof (first part, the second part is analogous):

Suppose that $\mathbf{\Sigma}^p_k=\mathbf{\Pi}^p_k$, and take $L\in\mathbf{\Sigma}^p_{k+1}$. Then L is recognized by a nondeterministic machine M with oracle for $L'\in\mathbf{\Sigma}^p_k=\mathbf{\Pi}^p_k$, and L' is recognized by a nondeterministic machine M_+ with oracle for $L_+\in\mathbf{\Sigma}^p_{k-1}$, and the complement of L' is recognized by a nondeterministic machine M_- with oracle for $L_-\in\mathbf{\Sigma}^p_{k-1}$. We can assume that both M_+ and M_- use the same oracle $L_\pm=\{(i,x) : x\in L_i\}\in\mathbf{\Sigma}^p_{k-1}$.

We modify machine M to a machine with oracle $L_\pm$ – instead of asking a query to L', it guesses an accepting run of M_+ or an accepting run of M_-. Thus $L\in\mathbf{\Sigma}^p_{k+1}$.

Other equalities follow easily.

Polynomial hierarchy

$$\begin{tikzcd}[column sep=small]
 & \mathbf{NP}=\Sigma_1^p \arrow[r, phantom, "\subseteq"] \arrow[ddr, phantom, "\subseteq"] & \Sigma_2^p \arrow[r, phantom, "\subseteq"] \arrow[ddr, phantom, "\subseteq"] & \Sigma_3^p \arrow[r, phantom, "\subseteq"] \arrow[ddr, phantom, "\subseteq"] & \cdots \\
\mathbf{P} \arrow[ur, phantom, "\subseteq"] \arrow[dr, phantom, "\subseteq"] & & & & \\
 & \mathbf{coNP}=\Pi_1^p \arrow[r, phantom, "\subseteq"] \arrow[uur, phantom, "\subseteq"] & \Pi_2^p \arrow[r, phantom, "\subseteq"] \arrow[uur, phantom, "\subseteq"] & \Pi_3^p \arrow[r, phantom, "\subseteq"] \arrow[uur, phantom, "\subseteq"] & \cdots
\end{tikzcd}$$

There are only two possibilities:

- either all the classes are different, or
- they are different to some point, and then they start to be equal

Complete language in Σ_k^p?

Input: a sentence of the following form (with *k* blocks of quantifiers)

$$\exists x_{11},\ldots,x_{1n}\forall x_{21},\ldots,x_{2n}\exists x_{21},\ldots,x_{2n} \ldots Qx_{k1},\ldots,x_{kn}\ \phi(x_{11},\ldots,x_{kn})$$

Question: is the sentence true?

Polynomial hierarchy

$$
\begin{array}{ccccccccc}
 & & \mathbf{NP}=\Sigma_1^p & \subseteq & \Sigma_2^p & \subseteq & \Sigma_3^p & \subseteq & \cdots \\
 & \rotatebox{45}{$\subseteq$} & & \rotatebox{-45}{$\subseteq$}\ \rotatebox{45}{$\subseteq$} & & \rotatebox{-45}{$\subseteq$}\ \rotatebox{45}{$\subseteq$} & & \rotatebox{-45}{$\subseteq$}\ \rotatebox{45}{$\subseteq$} & \\
\mathbf{P} & & & & & & & & \\
 & \rotatebox{-45}{$\subseteq$} & & & & & & & \\
 & & \mathbf{coNP}=\Pi_1^p & \subseteq & \Pi_2^p & \subseteq & \Pi_3^p & \subseteq & \cdots
\end{array}
$$

There are only two possibilities:

- either all the classes are different, or
- they are different to some point, and then they start to be equal

Complete language in Σ_k^p?

Input: a sentence of the following form (with k blocks of quantifiers)

$$\exists x_{11},\ldots,x_{1n} \forall x_{21},\ldots,x_{2n} \exists x_{21},\ldots,x_{2n} \ldots Q x_{k1},\ldots,x_{kn}\ \phi(x_{11},\ldots,x_{kn})$$

Question: is the sentence true? (similarly for Π_k^p)

Complete language in **PH**?

Fact 4:

If there exists a **PH**-complete language, then **PH**$=\Sigma_k^p$ for some k

Proof – The **PH**-complete language belongs to some Σ_k^p, and Σ_k^p is closed under reductions in polynomial time.

Polynomial hierarchy

Fact 2: $\Sigma^p_k \subseteq \Sigma^p_{k+1}$, $\Sigma^p_k \subseteq \Pi^p_{k+1}$, $\Pi^p_k \subseteq \Sigma^p_{k+1}$, $\Pi^p_k \subseteq \Pi^p_{k+1}$

Fact 3:

If $\Sigma^p_k = \Pi^p_k$, then $\Sigma^p_k = \Sigma^p_{k+1} = \ldots = \Pi^p_k = \Pi^p_{k+1} = \ldots = \mathbf{PH}$.

If **P**=**NP**, then $\mathbf{P} = \Sigma^p_1 = \Sigma^p_2 = \ldots = \Pi^p_1 = \Pi^p_2 = \ldots = \mathbf{PH}$.

Fact 4:

If there exists a **PH**-complete language, then $\mathbf{PH} = \Sigma^p_k$ for some k

Fact 5: **PH**⊆**PSPACE**

Proof: The Σ^p_k-complete language mentioned above is a special case of QBF, which belongs to **PSPACE**.

Polynomial hierarchy

Fact 2: $\Sigma_k^p \subseteq \Sigma_{k+1}^p$, $\Sigma_k^p \subseteq \Pi_{k+1}^p$, $\Pi_k^p \subseteq \Sigma_{k+1}^p$, $\Pi_k^p \subseteq \Pi_{k+1}^p$

Fact 3:

If $\Sigma_k^p = \Pi_k^p$, then $\Sigma_k^p = \Sigma_{k+1}^p = \ldots = \Pi_k^p = \Pi_{k+1}^p = \ldots =$ **PH**.

If **P**=**NP**, then **P**$=\Sigma_1^p = \Sigma_2^p = \ldots = \Pi_1^p = \Pi_2^p = \ldots =$**PH**.

Fact 4:

If there exists a **PH**-complete language, then **PH**$=\Sigma_k^p$ for some k

Fact 5: **PH**⊆**PSPACE**

Fact 6: If the classes Σ_k^p are all different, then **PH**≠**PSPACE**

Proof: Follows from Fact 4 – in **PSPACE** there is a complete language.

Alternating machines

- Alternating Turing machines (ATM) generalize nondeterministic ones (NTM)
- Both NTM and ATM are not a realistic model of computation (we cannot build such machines). But NTM help us to observe a very natural phenomenon: a difference between finding a solution and verifying a solution.
- ATMs have a similar role for some languages, for which there are no short witnesses, i.e., which cannot be characterized using nondeterminism.

Alternating machines

Definition of ATM:

- a configuration can have multiple successors (as in NTM)
- additionally states of the machine (and in effect its configurations) are divided to existential and universal ones

Alternating machines

Definition of ATM:

- a configuration can have multiple successors (as in NTM)
- additionally states of the machine (and in effect its configurations) are divided to existential and universal ones

The set of <u>wining</u> configurations is defined as the smallest set s.t.:

- accepting configurations are winning
- every <u>existential</u> configuration, whose <u>some</u> successor is winning, is also winning
- every <u>universal</u> configuration, whose <u>all</u> successors are winning, is also winning

We accept a word w, if the initial configuration for this word is winning.

M works in time $T(n)$ / in space $S(n)$, if every computation fits in this time / space.

Alternating machines

Definition of ATM:

- a configuration can have multiple successors (as in NTM)
- additionally states of the machine (and in effect its configurations) are divided to existential and universal ones

The set of <u>wining</u> configurations is defined as the smallest set s.t.:

- accepting configurations are winning
- every <u>existential</u> configuration, whose <u>some</u> successor is winning, is also winning
- every <u>universal</u> configuration, whose <u>all</u> successors are winning, is also winning

We accept a word w, if the initial configuration for this word is winning.

M works in time $T(n)$ / in space $S(n)$, if every computation fits in this time / space.

Observation:
NTM is a special case of an ATM – only existential states

Alternating machines

Equivalently: acceptance can be defined using a game:

- we consider the configuration graph (edges = possible transitions)
- players ∃ and ∀ alternatingly move a pawn (common to both player) around the graph
- in existential states player ∃ decides, in universal states player ∀ decides (player ∃ wants to accept, player ∀ wants to reject)
- we accept a word, if player ∃ has a winning strategy – he can reach an accepting configuration regardless moves of player ∀

Alternating machines

Classes **ATIME**($T(n)$), **ASPACE**($S(n)$),

AP=$\cup_k$ **ATIME**(n^k), **AL**=**ASPACE**($log\ n$)

Theorem

AL=**P**, **AP**=**PSPACE** (the same can be said more generally)

Alternating machines

Classes **ATIME**($T(n)$), **ASPACE**($S(n)$),

AP$=\cup_k$ **ATIME**(n^k), **AL**=**ASPACE**($log\ n$)

Theorem

AL=**P**, **AP**=**PSPACE** (the same can be said more generally)

Proof **AP**$\subseteq$**PSPACE**

Backtracking: we browse through all computations of the alternating machine (such a computation can be represented in polynomial space)

Alternating machines

Classes **ATIME**($T(n)$), **ASPACE**($S(n)$),

AP$=\cup_k$ **ATIME**(n^k), **AL**=**ASPACE**($log\ n$)

Theorem

AL=**P**, **AP**=**PSPACE** (the same can be said more generally)

Proof **AL**$\subseteq$**P**

We construct the graph containing all reachable configurations of the alternating machine – it is of polynomial size. Then in polynomial time we can find all winning configurations, by going backwards (starting from accepting configurations).

Alternating machines

Classes **ATIME**($T(n)$), **ASPACE**($S(n)$),

AP=$\cup_k$ **ATIME**(n^k), **AL**=**ASPACE**($log\ n$)

<u>Theorem</u>

AL=**P**, **AP**=**PSPACE** (the same can be said more generally)

<u>Proof</u> **PSPACE**⊆**AP**

It is enough to prove that QBF∈**AP**, as QBF is **PSPACE**-complete. This is almost obvious – player ∃ chooses values of variables quantified existentially, and player ∀ chooses values of variables quantified universally; at the end we deterministically compute the value of the formula.

Actually: the algorithm for **AP** is simpler than for **PSPACE**.

Alternating machines

Classes **ATIME**($T(n)$), **ASPACE**($S(n)$),

AP$=\cup_k$ **ATIME**(n^k), **AL**=**ASPACE**($\log n$)

Theorem

AL=**P**, **AP**=**PSPACE** (the same can be said more generally)

Proof **P**$\subseteq$**AL**

- For an algorithm in **P** there is an equivalent boolean circuit, and we can construct it in logarithmic space.
- It is easy to give an algorithm in **AL**, which computes the value of a circuit: players walk from the output gate, in OR gates player $\exists$ decides which predecessor is true, and in AND gates player $\forall$ decides which predecessor is supposed to be false.
- We do not generate the whole circuit, only particular fragments, „on demand".

Alternating machines

Consider alternating machines which:

- work in polynomial time
- the initial state is existential (universal)
- every computation leads to at most $k\text{-}1$ changes between existential and universal states

<u>Fact</u>

Such machines recognize languages from Σ^p_k (Π^p_k)

(we skip the formal proof, although it is easy)

Probabilistic machines

Machines with a source of random bits (probabilistic machines):

- a deterministic machine
- an additional read-once tape (the head cannot move left along this tape)

Probabilistic machines

Machines with a source of random bits (probabilistic machines):

- a deterministic machine
- an additional read-once tape (the head cannot move left along this tape)

Notice that **NP** can be defined as follows: a language L is in **NP** iff there is a polynomial $p(n)$ and a machine M with a source of random bits, working in at most $p(n)$ steps, and such that:

- $w \in L \Rightarrow \exists s.\ (w,s) \in L_M$
- $w \notin L \Rightarrow \nexists s.\ (w,s) \in L_M$

(a word is in L iff some witness confirms this)

Probabilistic machines

Machines with a source of random bits (probabilistic machines):

- a deterministic machine
- an additional read-once tape (the head cannot move left along this tape)

Notice that **NP** can be defined as follows: a language L is in **NP** iff there is a polynomial $p(n)$ and a machine M with a source of random bits, working in at most $p(n)$ steps, and such that:

- $w \in L \Rightarrow \exists s.\ (w,s) \in L_M$
- $w \notin L \Rightarrow \nexists s.\ (w,s) \in L_M$

Class **RP** (randomized polynomial time): as above, but

- $w \in L \Rightarrow Pr_s[(w,s) \in L_M] \geq 0.5$
- $w \notin L \Rightarrow \nexists s.\ (w,s) \in L_M$

Intuition: a word is in L, if at least half of possible witnesses confirm this.

Probabilistic machines

Class **RP** (randomized polynomial time): a language L is in **RP** iff there is a polynomial $p(n)$ and a machine M with a source of random bits, working in at most $p(n)$ steps, and such that:

- $w \in L \Rightarrow Pr_s[(w,s) \in L_M] \geq 0.5$
- $w \notin L \Rightarrow \nexists s.\ (w,s) \in L_M$

As s we can take sequences of length $p(n)$, or infinite sequences, does not matter.

Intuition: a word is in L, if at least half of possible witnesses confirm this (but there are no witnesses for words not in L)

In other words: if a word is not in L, we will certainly reject;
if it is in L, then choosing transitions randomly, we will accept with probability at least 0.5

Probabilistic machines

Class **RP** (randomized polynomial time): a language L is in **RP** iff there is a polynomial $p(n)$ and a machine M with a source of random bits, working in at most $p(n)$ steps, and such that:

- $w \in L \Rightarrow Pr_s[(w,s) \in L_M] \geq 0.5$
- $w \notin L \Rightarrow \nexists s.\ (w,s) \in L_M$

Remark: Some machines does not accept any language in the sense of **RP**. It is undecidable whether a machine is correct in the sense of **RP**, even if we know the polynomial $p(n)$

For this reason we do not know any **RP**-complete problem.
Intuition: we cannot reduce from every machine recognizing a language from **RP**, because we do not know how such machines look like.

Probabilistic machines

Class **RP** (randomized polynomial time): a language L is in **RP** iff there is a polynomial $p(n)$ and a machine M with a source of random bits, working in at most $p(n)$ steps, and such that:

- $w \in L \Rightarrow Pr_s[(w,s) \in L_M] \geq 0.5$
- $w \notin L \Rightarrow \nexists s.\ (w,s) \in L_M$

<u>Fact</u>: **P**$\subseteq$**RP**$\subseteq$**NP** (both inclusions are obvious)

Probabilistic machines

Class **RP** (randomized polynomial time): a language L is in **RP** iff there is a polynomial $T(n)$ and a machine M with a source of random bits, working in at most $T(n)$ steps, and such that:

- $w \in L \Rightarrow Pr_s[(w,s) \in L_M] \geq 1\text{-}p=0.5$
- $w \notin L \Rightarrow \nexists s.\ (w,s) \in L_M$

<u>Fact</u> (amplification): in the definition of **RP** the number 0.5 can be changed to any number from the interval (0,1), and the class of defined languages will remain the same

Proof: Let $\mathbf{RP}_p$ be the class with <u>error</u> probability p

Obviously $\mathbf{RP}_p \subseteq \mathbf{RP}_q$ when $p \leq q$

We will now prove that $\mathbf{RP}_p \subseteq \mathbf{RP}_{p^2}$

- Out of a machine M with error p we construct a machine M', which on the same input chooses randomly two witnesses, and accepts if some of them is a correct witness
- The running time doubles, so it remains polynomial
- The error probability decreases to p^2 – M' is wrong only when M made a mistake twice

Probabilistic machines

Is this a realistic model?

- It is more realistic than nondeterministic or alternating machines: we can run a probabilistic machine, give it some sequence of bits as random bits, and obtain a result that is correct with some probability.
- We obtain a result that is correct with some probability (and due to amplification this probability can be arbitrarily high), but we cannot be sure.
- How to generate bits that are really random? There exist physical random number generators (basing e.g. on quantum effects). Problems: they are relatively slow, and can be biased (in particular after some time, when they start to be broken).
- In practice, we use pseudo-random generators, that generate “random” bits using some algorithm. In practice, this works well, as the generated sequence looks like a random one.
  But theoretically, we cannot be sure about the probability of correctness.