1 Vztahy mezi složitostními třídami

Již jsme obeznámeni s následujícími skutečnostmi:

- Je možné dosáhnout konstantního zrychlení za cenu nárůstu potřebné paměti.
- Determinismus je speciálním případem nedeterminismu:

$$\mathsf{DX}(f) \subseteq \mathsf{NX}(f)$$

- Platí následující vztah mezi časem a prostorem:

$$\mathsf{XTIME}(T) \subseteq \mathsf{XSPACE}(T)$$
$$\mathsf{XSPACE}(S) \subseteq \mathsf{XTIME}(2^{O(S)})$$

- Pro $S(n) \geq \log n$ platí

$$\mathsf{NTIME}(T(n)) \subseteq \mathsf{DSPACE}(T(n))$$
$$\mathsf{NSPACE}(S(n)) \subseteq \mathsf{DTIME}(2^{O(S(n))})$$

  Idea důkazu: převod NTS na DTS prostřednictvím simulaci každé konfigurace a následní procházení grafem, jehož velikost odpovídá počtu konfigurací.

- Savitchova věta:

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

  Konstrukce používá pro simulaci rekurzivní proceduru pro hledání cesty (výpočtu) mezi dvěma konfiguracemi simulovaného stroje.

- $$\mathsf{DLOG} \subseteq \mathsf{NLOG} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq$$
$$\mathsf{EXPTIME} \subseteq \mathsf{NEXPTIME} \subseteq \mathsf{EXPSPACE} \ldots$$

1.1 Uzavřenost složitostních tříd

Deterministické složitostní třídy jsou uzavřeny na $+, \cdot, \cap, \cup$, komplement, ... Komplementem třídy $\mathcal{C}$ se rozumí třída $co-\mathcal{C} = \{co-L | L \in \mathcal{C}\}$. U nedeterministických tříd je uzavřenost na komplement otevřeným problémem.

1.2 Immerman – Szelepcsénejiho věta

Věta 1.1. *Nechť* $S(n) \geq \log n$. *Pak* $\mathsf{NSPACE}(S(n) = co - \mathsf{NSPACE}(S(n))$.

Důkaz. Idea důkazu: Předpokládejme, že A je konečná množina řetězců. Známe velikost A a nedeterministický test příslušnosti do A (T_P). Z toho je možné vytvořit test T_N pro nepříslušnost do A:

1. Nedeterministicky uhádmene $|A|$ řetězců.
2. Pro každé zvolené α nedeterministicky ověřím, zda $\alpha \in A$.
3. Pokud všechny zvolené řetězce patří do A a x není mezi nimi, platí $x \notin A$.

Důkaz: Je daný NTS M s prostorovou složitostí $S(n)$. Konfiguraci M reprezentuje řetězec nad Δ. Nechť $|\Delta| = d$. Pro vstup délky n existuje $d^{S(n)}$ konfigurací a každá konfigurace $C \in \Delta^{S(n)}$. Lze předpokládat, že existuje právě jedna akceptující konfigurace pro vstupy délky n ($accept \in \Delta^{S(n)}$). Označme $start \in \Delta^{S(n)}$ počáteční konfiguraci.

Uvažme vstupní řetězec x, kde $|x| = n$. Počet konfigurací dosažitelných při výpočtu na x je nejvýše $d^{S(n)}$. Označme

$$A_m = \{\alpha \in \Delta^{S(n)} | start \to \alpha \text{ v nejvýše } m \text{ krocích}\}$$
$$A_0 = \{start\}$$

Platí

$$M \text{ akceptuje } x \iff accept \in A_{d^{S(n)}}$$

Nyní můžeme přistoupit ke konstrukci NTS N s prostorovou složitostí $S(n)$, takového, že $L(N) = co - L(M)$:

1. Pro vstup délky n na pracovní pásce označí $S(n)$ políček. Předpokládáme, že $S(n)$ je prostově konstruovatelná.
2. Induktivně počítá $|A_0|, |A_1|, \ldots, |A_{d^{S(n)}}|$:
   (a) $|A_0| = 1$
   (b) Známe $|A_m|$, počítáme $|A_{m+1}|$ tak, že v lexikografickém pořadí testujeme všechny řetězce $\beta \in \Delta^{S(n)}$, zda patří do A_{m+1}. Pokud ano, zvýšíme počitadlo.

   Test, zda $\beta \in A_{m+1}$ probíhá takto:

   - Nedeterministicky zvolíme $|A_m|$ řetězců pro každé zvolené α.

- Nedeterministicky ověříme, zda $\alpha \in A_m$.
- Deterministicky ověříme, zda $\alpha \to \beta$ v nejvýše jednom kroku.

3. Po výpočtu $|A_{d^{S(n)}}|$ otestujeme, zda $accept \in A_{d^{S(n)}}$. Pokud ano, N zamítá, jinak akceptuje.

   Pro test, zda $accept \in A_{d^{S(n)}}$ nedeterministicky zvolíme $|A_{d^{S(n)}}|$. Pro každý takový řetězec α ověříme $start \to \alpha$ v nejvýše $d^{S(n)}$ krocích a když žádný z nich nebyl „nový“ $accept$, tak $accept \notin A_{d^{S(n)}}$.

Není-li $S(n)$ prostorově konstruovatelná, označíme na začátku jen jedno políčko. Jakmile jsou při výpočtu přesažena označená políčka, označíme další políčko a restartujeme výpočet. □

1.3 Separace složitostních tříd

V následující části se pokusíme najít odpovědi na otázky typu „je hierarchie jazyků z hlediska složitosti nekonečná?“, „existuje horní mez složitosti pro všechny jazyky?“, „jak velký musí být rozdíl mezi funkcemi f a g, aby byly třídy $[d+n][time+space]([f+g](n))$ různé?“

1.3.1 DSPACE

Věta 1.2. *Nechť $S(n)$ je prostorově konstruovatelná funkce. Potom existuje jazyk z $\mathsf{DSPACE}(S(n))$, který nepatří do $\mathsf{DSPACE}(S'(n))$ pro žádné $S'(n) \in o(S(n))$.*

Důkaz. Pro dané $S(n)$ hledáme nějaký jazyk L splňující podmínky věty. K jeho nalezení použijeme diagonalizaci.

Mějme seznam $M_0, M_1, M_2, \ldots$ DTS. Každý DTS lze binárně zakódovat tak, že M_i má kód i a každé binární číslo kóduje nějaký DTS. Vedoucí nuly v binárním řetězci ingorujeme ($00101 \approx 101$). Označme $\sharp(x)$ číslo reprezentované řetězcem x. Pro všechna dostatečně velká n existuje binární řetězec x délky n takový, že $\sharp(x) = i$ – toto platí právě díky možnosti doplnit vedoucí nuly.

Nyní zkonstruujeme DTS M takový, že $L(M) \in \mathsf{DSPACE}(S(n))$, ale $L(m) \notin \mathsf{DSPACE}(S'(n))$. Pro vstup délky n:

1. označí $S(n)$ políček pracovní pásky,
2. simuluje na označených políčkách stroj M_i na x, kde $i = \sharp(x)$.

Nastane právě jedna možnost:

i. M úspěšně dokončí simulaci M_i na x. Pak M akceptuje $\iff$ M_i zamítá.

ii. Pokud M_i cyklí, tak i M cyklí.

iii. Pokud M_i použije více než $S(n)$ políček, tak M zamítá.

Nyní ukážeme $L(M) \neq L(M')$. označme $q(n)$ prostorovou složitost M'.

- $q(n) \gg S(n)$ – takové dts se v seznamu mohou vyskytovat, krátký kód stroje nemůže zaručit „malou“ prostorovou složitost. V tomto případě se jazyky nemusí lišit, ale nevadí to, protože chceme ukázat pouze neexistenci jazyků s nižší prostorovou složitostí.

- $q(n) \in o(S(n))$ – je třeba si uvědomit, že M má předem pevně danou pracovní abecedu. M' přitom může mít větší pracovní abecedu. Je proto nezbytné použít (binární) kódování pracovních symbolů M'. To přináší konstantní nárůst složitosti. Připomeňme:

  $$q(n) \in o(S(n)) \iff \forall d \exists n_0 \forall n > n_0 . q(n) < d \cdot S(n)$$

  Proto dostatek prostoru pro pracovní abecedu máme až od jistého n_0, což ale může přinášet potíže při simulaci výpočtu na menších slovech. To ovšem není na závadu díky skutečnosti, že každý stroj se v tabulce vyskytuje mnohokrát. Některý jeho kód je již dostatečně dlouhý. Jinými slovy daný stroj se nemusí „vydiagonalizovat“ hned při prvním výskytu v tabulce.

Celkem lze nahlédnout, že M má prostorovou složitost $S(n)$ a neexistuje DTS s asymtoticky nižší prostorovou složitostí pro jazyk $L(m)$. □

1.3.2 DTIME

Věta 1.3. *Nechť $T(n)$ je časově konstruovatelná funkce. pak existuje jazyk $L \in \mathsf{DTIME}(T(n))$, který nepatří do $\mathsf{DTIME}(T'(n))$ pro žádné $T'(n)$ takové, že*

$$T'(n) \log T'(n) \in o(T(n)).$$

Důkaz. Důkaz je obdobný jako v případě DSPACE, ovšem simulaci stroje již nelze provést bez zpomalení. □

1.3.3 NSPACE

Výše uvedenou konstrukci pro DSPACE nelze jednoduše převzít, protože záměna akceptujících a zamítajících stavů u nedeterministických strojů nefunguje. Používá se *vycpávací technika (padding technique)*. Techniku nebudeme ukazovat v obecné podobě, nýbrž na konkrétním příkladu.

Věta 1.4. $\mathsf{NSPACE}(n^3) \subset \mathsf{NSPACE}(n^4)$

Důkaz. Sporem. Nechť $\mathsf{NSPACE}(n^4) \subseteq \mathsf{NSPACE}(n^3)$. Stejným způsobem postupně ukážeme $\mathsf{NSPACE}(n^7) \subseteq \mathsf{NSPACE}(n^6) \subseteq \mathsf{NSPACE}(n^5) \subseteq \mathsf{NSPACE}(n^4)$. Celkem dostaneme

$$\mathsf{NSPACE}(n^7) \subseteq \mathsf{NSPACE}(n^3) \subseteq \mathsf{DSPACE}(n^6)(\text{Savitchova věta}) \subset$$
$$\subset \mathsf{DSPACE}(n^7)(\text{věta 1.2}) \subseteq \mathsf{NSPACE}(n^7),$$

a tedy spor.

Pro začátek chceme $\mathsf{NSPACE}(n^5) \subseteq \mathsf{NSPACE}(n^4)$. Mějme jazyk $A \in \mathsf{NSPACE}(n^5)$ a NTS M takový, že $L(M) = A$ a M je n^5-prostorově ohraničen. Nechť

$$A' = \{x\sharp^{|x|^{\frac{5}{4}}-|x|} | x \in A\}$$

Nechť M' je NTS pro jazyk A'. M' pro vstup $x\sharp^m$ provede následující:

i. Ověří, zda $m = |x|^{\frac{5}{4}} - |x|$.

ii. Pokud ano, tak simuluje M na x.

Prostorová složitost je

$$|x|^5 = (|x|^{\frac{5}{4}})^4 = |x\sharp^{|x|^{\frac{5}{4}}-|x|}|^4,$$

takže $A' = L(M') \in \mathsf{NSPACE}(n^4)$. Proto také $A' \in \mathsf{NSPACE}(n^3)$ (Předpokládáme, že $\mathsf{NSPACE}(n^4) \subseteq \mathsf{NSPACE}(n^3)$. Nechť M'' je NTS s prostorovou složitostí n^3 takový, že $A' = L(M'')$. Nyní zkonstruujeme NTS M''' pro A:

i. Za vstupní řetězec x připojí $\sharp^{|x|^{\frac{5}{4}}-|x|}$.

ii. Simuluje M''.

Analyzujme nyní prostorovou složitost stroje M'''.

$$n^3 = |x\sharp^{|x|^{\frac{5}{4}}-|x|}|^3 = (|x|^{\frac{5}{4}})^3 = |x|^{\frac{15}{4}} \leq |x|^4$$

Proto $A = L(M''') \in \mathsf{NSPACE}(n^4)$.

Dále postupujeme analogicky pro $\mathsf{NSPACE}(n^6)$ a $\mathsf{NSPACE}(n^7)$. Nakonec dostáváme spor s pomocí konstrukce uvedené na začátku důkazu. □

Poznámka. Tvrzení o separaci NSPACE je také důsledkem lematu o uzavřenosti N-tříd na komplement.

1.4 PSPACE-úplnost

K tomuto tématu nejsou bohužel k dispozici žádné podklady a na přednášce jsem nebyl. Jako PSPACE-úplný problém byl definován QBF – Quantified Boolean Formula, tedy formule predikátové logiky. Jiným pohledem na QBF je potom hra dvou hráčů. Jeden hráč ($\forall$) se snaží zvítězit tak, že vybírá univerzálně kvantifikované proměnné tak, aby hráč $\exists$ nemohl zvolit žádný použitelný existenčně kvantifikovaný prvek.

1.5 NLOG-úplnost

Vlastnosti relace $\leq_{\log}$ (redukce v logaritmickém prostoru):

1. Tranzitivita: $A \leq_{\log} B, B \leq_{\log} C \Rightarrow A \leq_{\log} C$
2. $A \subseteq \Sigma^\star, A \in \mathsf{DLOG} \Rightarrow A \leq_{\log} \{0,1\}$
3. $A \leq_{\log} B, B \in \mathsf{DLOG} \iff A \in \mathsf{DLOG}$
4. $A \leq_{\log} B \Rightarrow A \leq_{\text{poly}} B$, kde $\leq_{\text{poly}}$ označuje redukci v polynomiálním *čase*. Obrácená implikace je otevřeným problémem.

Věta 1.5. *Problém dosažitelnosti v orientovaném grafu (MAZE) je* NLOG*-úplný. MAZE lze formálně zavést takto:* $MAZE = \{\langle G, s, t\rangle |$ *v grafu* G *existuje cesta z* s *do* $t\}$.

Důkaz.

- $MAZE \in \mathsf{NLOG}$: Z vrcholu s nedeterministicky vyberu hranu do některého následníka, vymažu z pásky s, zapíši následníka a pokračuji, dokud nedosáhnu vrcholu t nebo počet kroků nepřekročí $|V|$. Zápis identifikace vrcholu i počitadlo kroků lze realizovat v logaritmickém prostoru.
- $A \in \mathsf{NLOG} \Rightarrow A \leq_{\log} MAZE$: $w \to R(w), R(w) = G(V,E)$, kde V obsahuje všechny konfigurace výpočtů na vstupu délky $|w|$ a E odpovídá kroku výpočtu (relaci $\models$). Položíme-li $s =$ počáteční konfigurace výpočtu a $t =$ akceptující konfigurace, platí $w \in A \iff R(w) \in MAZE$. Lze nahlédnout, že G lze vypočítat v logaritmickém prostoru.

□

Věta 1.6. $MAZE \in \mathsf{DLOG} \iff \mathsf{NLOG} = \mathsf{DLOG}$

Poznámka. Další NLOG-úplné problémy:

- Pro daný NFA zjistit, zda akceptuje neprázdný jazyk.
- Pro daný NFA zjistit, zda akceptuje nekonečný jazyk.

1.6 P-úplnost

Zkoumání P-úplnosti má význam pro řešení problémů, zda $\mathsf{P} = \mathsf{NLOG}$ resp. $\mathsf{P} = \mathsf{DLOG}$. P-úplnost budeme zkoumat vzhledem k relaci $\leq_{\log}$. Vzhledem k polynomiální časové redukci jsou totiž všechny jazyky kromě $\Sigma^\star$ a $\emptyset$ P-úplné.

1.6.1 Circuit Value Problem (CVP)

Logický obvod je program obsahující konečný počet přiřazení tvaru

$$\begin{aligned} P_i &= 0 \\ P_i &= 1 \\ P_i &= P_j \wedge P_k, j, k \ < i \\ P_i &= P_j \vee P_k, j, k \ < i \\ P_i &= \neg P_j, j < i, \end{aligned}$$

kde každé P_i na levé straně vyskytuje právě jednou. Úlohou je vypočítat hodnotu P_n, kde n je nejvyšší index.

Věta 1.7. *CVP je $\leq_{\log}$-úplný problém pro* P.

Důkaz.

- $CVP \in \mathsf{P}$: zřejmé.
- $A \in \mathsf{P} \Rightarrow A \leq_{\log} CVP$: Uvažme jazyk A a jednopáskový DTS M takový, že $L(M) = A$. Lze předpokládat, že časová složitost M je n^c.

  Pro vstup x délky n zkonstruujeme tabulku velikosti $(n^c+1) \times (n^c+1)$, kde i-tý řádek má tvar

$\triangleright$	a	b	a	a	b	a	b	$\sqcup$	$\sqcup$
						p			

  a $(i+1)$. řádek

$\triangleright$	a	b	a	a	b	b	b	$\sqcup$	$\sqcup$
					q				

  pokud $\delta(p, a) = (q, b, L)$. Řádek tedy zachycuje jednu konfiguraci výpočtu a posloupnost sloupců odpovídá přechodům mezi konfiguracemi (výpočtu).

  Tabulka dává základ pro vytvoření CVP odpovídajícího DTS M. Budou použity proměnné následujících tvarů:

– $P^a_{ij}, 0 \leq i,j \leq n^c, a \in \Gamma$ – *true* $\approx$ tabulka má na pozici i,j symbol a

– $Q^q_{ij}, 0 \leq i,j \leq n^c, , q \in Q$ – *true* $\approx$ tabulka má na pozici i,j stav q

Přiřazení logického obvodu: $\forall 1 \leq i \leq n^c, 0 \leq j \leq n^c, b \in \Gamma$

$$P^b_{ij} = \bigvee_{\delta(p,a)=(q,b,d)} (Q^p_{i-1,j} \wedge P^a_{i-1,j}) \vee (P^b_{i-1,j} \wedge \bigwedge_{p \in Q} \neg Q^p_{i-1,j})$$

Výraz lze přečíst jako „na j-tém políčku pásky je po i-tém kroku znak b, když před i-tým krokem byla hlava na j-tém políčku a v i-tém kroku bylo zapsáno b, nebo pokud byla před i-tým krokem hlava na jiném políčku a symbol b zde byl i předtím“.

$\forall 1 \leq i \leq n^c, 1 \leq j \leq n^c - 1, q \in Q$

$$Q^q_{ij} = \bigvee_{\delta(p,a)=(q,b,R)} (Q^p_{i-1,j-1} \wedge P^a_{i-1,j-1}) \vee \bigvee_{\delta(p,a)=(q,b,L)} (Q^p_{i-1,j+1} \wedge P^a_{i-1,j+1})$$

Pro $j = 0$ zapíšeme jen první část výrazu, pro $j = n^c$ jen druhou část.

Zbývá zakódování počáteční konfigurace. Nechť $x = a_1 a_2 \dots a_n$.

$$\begin{aligned}
P^{\triangleright}_{00} &= 1 \\
P^b_{00} &= 0, b \in \Gamma \setminus \{\triangleright\} \\
P^{a_j}_{0,j} &= 1, 1 \leq j \leq n \\
P^b_{0,j} &= 0, 1 \leq j \leq n, b \in \Gamma \setminus \{a_j\} \\
P^{\sqcup}_{0,j} &= 1, n+1 \leq j \leq n^c \\
P^b_{0,j} &= 0, n+1 \leq j \leq n^c, b \in \Gamma \setminus \{\sqcup\}
\end{aligned}$$

$$\begin{aligned}
Q^s_{00} &= 1 \\
Q^s_{0,j} &= 0, 1 \leq j \leq n^c \\
Q^q_{0,j} &= 0, 0 \leq j \leq n^c, q \in Q \setminus \{s\}
\end{aligned}$$

Akceptování odpovídá hodnotě logického obvodu, přesněji proměnné $Q^t_{n^c,0}$, kde t je akceptující stav. Konstrukce logického obvodu je realizovatelná v prostoru $O(\log n)$.

□

Poznámka.

CVP – P-úplný

SAT – NP-úplný

QBF – PSPACE-úplný

1.7 Turingova redukce (T-redukce)

Jedná se o zobecnění běžné redukce – při výpočtu jednoho problému můžeme volat algoritmus pro druhý problém nikoliv jen jednou, nýbrž vícekrát (s různými parametry).

Definice 1.8. Problém A *není algoritmicky těžší* než problém B, pokud existuje algoritmus pro A, který může využívat algoritmus pro B a má následující vlastnosti:

- Časová složitost algoritmu pro A bez volání algoritmu pro B je polynomiální ($\leq p(n)$).
- Počet volání algoritmu pro B je polynomiální ($\leq q(n)$).
- Délka argumentu, pro který se volá algoritmus pro B, je polynomiální ($\leq r(n)$).

Označme $t_B(n)$ časovou složitost algoritmu pro B. Potom A můžeme řešit v čase

$$t_A(n) \leq p(n) + q(n) \cdot t_B(r(n)).$$

Píšeme $A \leq_T B$. Dále $A \equiv_T B \iff A \leq_T B \wedge B \leq_T A$.

Poznámka. $\leq_T$ je tranzitivní.

Věta 1.9. *$TSP_{OPT} \equiv_T TSP_{EVAL} \equiv_T TSP_{DEC}$, kde TSP_{OPT} je algoritmus hledající řešení TSP, TSP_{EVAL} je algoritmus hledající cenu optimálního řešení TSP a TSP_{DEC} pro $\langle G, k \rangle$ rozhoduje, zda v G existuje řešení TSP s cenou nejvýše k.*

Důkaz.

- $TSP_{DEC} \leq_T TSP_{EVAL}, TSP_{EVAL} \leq_T TSP_{OPT}$ – triviální.
- $TSP_{OPT} \leq_T TSP_{EVAL}$ – Pro graf G nejprve algoritmem EVAL vypočítáme cenu řešení. Pak postupně odstraňujeme hrany z G. Po každé odstraněné hraně znovu počítáme cenu řešení. Pokud po odstranění nějaké hrany cena vzrostla, jedná se o hranu, která je součástí řešení, a proto ji vrátíme zpět a odstraníme jinou. Po otestování všech hran zůstává právě optimální řešení TSP.
- $TSP_{EVAL} \leq_T TSP_{DEC}$ – Horní odhad ceny řešení je součet ohodnocení všech hran v grafu. Začneme tedy s k rovným této hodnotě a opakovaným voláním DEC s půlením intervalu se dostaneme k ceně optimálního řešení.

□

2 Výpočetní modely

Poznámka. Dále probírané modely jsou abstraktní. Není možné je sestrojit, ani naprogramovat.

2.1 Orákulové Turingovy stroje

Definice 2.1. *Turingův stroj s orákulem B* (zkráceně OTS) je Turingův stroj s následujícími vlastnostmi:

- Má navíc jednu speciální pásku pouze pro zápis (*orákulovou pásku*, query tape), na kterou může zapisovat konečné řetězce.
- Řídicí jednotka má tři speciální stavy – $q_?, q_+, q_-$.
- Přechod TS do stavu $q_?$ je interpretován jako dotaz na řetězec y aktuálně zapsaný na orákulové pásce. Pokud $y \in B$, stroj přejde do q_+, pokud $y \notin B$, přejde do q_-. V obou případech vymaže obsah orákulové pásky.

Nechť M je OTS. Jazyk stroje M je nutné vztahovat ke konkrétnímu orákulu A. Pak píšeme $L(M(A))$.

Definice 2.2. Nechť A je jazyk. Pak $\mathsf{P}^A = \{L(M(A)) | M$ je ODTS s polynomiální časovou složitostí$\}$. Analogicky pro NP^A a další složitostní třídy. Pojem je možné rozšířit rovněž pro třídu jazyků C. Definujeme $\mathsf{P}^C = \bigcup_{A \in C} \mathsf{P}^A$.

Příklad.

$$\begin{aligned} \mathsf{P}^\emptyset &= \mathsf{P} \\ \mathsf{P}^{SAT} &= \mathsf{NP} \\ \mathsf{P}^\mathsf{P} &= \mathsf{P} \\ \mathsf{NP}^\mathsf{NP} &= ? \end{aligned}$$

Lema 2.3. *Je-li jazyk B úplný (ve smyslu redukce v polynomiálním čase) pro třídu jazyků C, platí $\mathsf{P}^B = \mathsf{P}^C$.*

Věta 2.4. *Existují orákula A, B taková, že*

1. $\mathsf{P}^A \neq \mathsf{NP}^A$
2. $\mathsf{P}^B = \mathsf{NP}^B$

Důkaz.

1. Ukážeme konstrukci orákula A požadovaných vlastností. Pro libovolné orákulum A položme

$$L_A = \{w | \exists x \in A. |x| = |w|\}$$

L_A je zřejmě v NP^A – pro vstup w stroj uhádne slovo délky $|w|$, které leží v A.

Zbývá dokázat $L_A \notin \mathsf{P}^A$. Orákulum A budeme postupně budovat v jednotlivých etapách. Nechť $M_1, M_2, \ldots$ je seznam všech OTS s polynomiální časovou složitostí. Pro jednoduchost uvažujme, že M_i má časovou složitost n^i. (To není na újmu obecnosti, neboť při standardní enumeraci TS se každý stroj objevuje v seznamu nekonečně mnohokrát.)

Etapa i: V etapě i zajistíme, že M_i^A nerozhoduje L_A pro konečný počet řetězců.

Vybereme n tak velké, bylo větší než je délka každého řetězce, jehož příslušnost do A již byla rozhodnuta a aby $2^n > n^i$ (n^i je časová složitost M_i. Nyní ukážeme, jak rozšířit doposud získanou informaci o A tak, aby M_i^A akceptoval 1^n, pokud tento řetězec nepatří do L_A.

Spustíme M_i na vstupu 1^n a na orákulové dotazy odpovídáme následujícím způsobem. Ptá-li se M_i na řetězec y, jehož příslušnost do A již byla určena (v některé předcházející etapě), odpovíme ve shodě s tímto určením. Nebyla-li příslušnost y dosud určena, odpovíme záporně a y určíme coby nepatřící do A. Pokračujeme, dokud M_i nezastaví.

Akceptuje-li M_i řetězec 1^n, nastavíme všem slovům délky n, že nenáleží do A. To lze provést, neboť n je dostatečně velké, a o příslušnosti slov délky n tedy ještě nebylo rozhodnuto. Protože A neobsahuje žádné slovo délky n, znamená to, že $1^n \notin L_A$, a tedy stroj M_i nemůže počítat L_A.

Pokud naopak M_i zamítá řetězec 1^n, potřebujeme (pro spor) zajistit $1^n \in L_A$, tedy potřebujeme zařadit nějaké slovo délky n do A. Stroj M_i se za dobu svého běhu n^i mohl zeptat nejvýše na n^i slov (ta byla označena jako nepatřící do A), ovšem protože $2^n > n^i$, stále zbývá alespoň jedno slovo, které můžeme do A zařadit.

Po nekonečném počtu etap je A takové orákulum, že $L_A \notin \mathsf{P}^A$.

2. Nechť $B = QBF$. Protože $QBF \in \mathsf{PSPACE}$, je možné odpověď orákula QBF vypočítat v deterministickém polynomiálním prostoru, a tedy $\mathsf{NP}^{QBF} \subseteq \mathsf{NPSPACE}$. Ze Savitchovy věty dostáváme $\mathsf{NPSPACE} = \mathsf{PSPACE}$. Předcházející lemma dává $\mathsf{PSPACE} = \mathsf{P}^{QBF}$. Celkem tedy

$$\mathsf{NP}^{QBF} \subseteq \mathsf{NPSPACE} = \mathsf{PSPACE} \subseteq \mathsf{P}^{QBF}$$

Inkluze $\mathsf{P}^{QBF} \subseteq \mathsf{NP}^{QBF}$ je zřejmá, celkem tedy $\mathsf{P}^{QBF} = \mathsf{NP}^{QBF}$.

□

Poznámka. $\mathsf{NSPACE}^A(f) \subseteq \mathsf{DSPACE}^A(f^2)$ – *techniku simulace* lze u orákulových strojů použít stejně jako u běžných Turingových strojů.

$\mathsf{DSPACE}^A(f) \subsetneq \mathsf{DSPACE}^A(g)$, kde $f \in o(g)$ – separace diagonalizací je u Turingových strojů rovněž použitelná.

Důsledek 2.5. $\mathsf{NP} = \mathsf{P}$ *není možno dokázat simulací, neboť* $\mathsf{NP} \subseteq \mathsf{P} \Rightarrow \forall A.\mathsf{NP}^A \subseteq \mathsf{P}^A$*, spor s větou 2.4.*

Rovněž tak technika diagonalizace není dost silná pro rozlišení P *a* NP*.*

Věta 2.6.

$$\mathcal{P}(\mathsf{P}^A \neq \mathsf{NP}^A) = 1$$

Poznámka.

$$\exists C_1, C_2.\mathcal{P}(C_1^A \neq C_2^A) = 1 \wedge C_1 = C_2$$

2.1.1 Polynomiální hierarchie

Definice 2.7 (Polynomiální hierarchie)**.** Definujeme systém tříd Σ_i^{P} a Π_i^{P} takto

$$\begin{aligned}
&\Sigma_0^{\mathsf{P}} = \mathsf{P}, \\
&\Sigma_{i+1}^{\mathsf{P}} = \mathsf{NP}^{\Sigma_i^{\mathsf{P}}}, \\
&\Pi_i^{\mathsf{P}} = co - \Sigma_i^{\mathsf{P}}.
\end{aligned}$$

Dále definujeme třídu PH,

$$\mathsf{PH} = \bigcup_{i=0}^{\infty} \Sigma_i^{\mathsf{P}} \cup \Pi_i^{\mathsf{P}}.$$

Definice 2.8. Relace R je polynomiálně vyvážená, právě když $\exists k.(x, y) \in R \Rightarrow |y| \leq |x|^k$. R je polynomiálně ověřitelná, když umíme v polynomiálním čase rozhodnout, zda dvojice řetězců je v relaci (nebo-li $R \in P$).

Poznámka (Alternativní charakterizace)**.** Budeme používat binární relaci $R \subseteq \Sigma^* \times \Sigma^*$.

Věta 2.9 (Charakterizace NP polynomiálním svědkem)**.** $L \in \mathsf{NP}$*, právě když existuje polynomiálně vyvážená a polynomiálně ověřitelná relace* R *taková, že* $L = \{x \mid \exists y.(x, y) \in R\}$*.*

Důkaz. $\Leftarrow$: Máme R, potřebujeme $L \in \mathsf{NP}$. Sestrojíme nedeterministický Turingův stroj, který pro dané x uhodne y a ověří, že $(x, y) \in R$.

$\Rightarrow$: Máme $L \in \mathsf{NP}$, a tedy nedeterministický stroj, který ho rozhoduje. Pak definujeme

$$R = \{(x, y) \mid y \text{ je akceptující výpočet na } x\}.$$

□

Poznámka. Podobně můžeme charakterizovat i třídu coNP: jazyk $L \in \mathsf{coNP}$, právě když existuje polynomiálně vyvážená a ověřitelná R tak, že

$$L = \{x \mid \forall y.|y| \leq |x|^k.(x, y) \in R\}.$$

Věta 2.10. *Jazyk $L \in \Sigma_i^{\mathsf{P}}$, právě když existuje polynomiálně vyvážená relace R tak, že*

$$R \in \Pi_{i-1}^{\mathsf{P}},$$
$$L = \{x \mid \exists y.(x, y) \in R\}.$$

Důkaz. Indukcí přes i.

$\Leftarrow$: Pro $i = 1$ tvrzení platí podle věty 2.9. Nechť tedy $i > 1$. Předpokládejme, že relace R existuje. Popíšeme nedeterministický orákulový Turingův stroj s orákulem z $\Sigma_{i-1}^{\mathsf{P}}$, který rozhoduje L. Stroj pro vstup x uhádne y a zeptá se orákula, zda $(x, y) \in R$. Protože $R \in \Pi_{i-1}^P$ a nikoliv v Σ_{i-1}^P, stroj odpověď orákula zneguje.

$\Rightarrow$: Máme $L \in \Sigma_i^{\mathsf{P}}$ a konstruujeme relaci R. Víme, že existuje orákulový nedeterministický Turingův stroj $M^?$, který rozhoduje L s orákulem $K \in \Sigma_{i-1}$. Na K použijeme indukční předpoklad: existuje relace $S \in \Pi_{i-2}$ taková, že $z \in K \Leftrightarrow \exists w.(z, w) \in S$.

Konstrukce R. Pro x bude certifikátem y akceptující výpočet M^K na x. Relace R musí být rozhodnutelná (musí být možné zkontrolovat, že y je akceptující výpočet). Jediný problém při kontrole je, když stroj provede dotaz na orakulum. Pro každý kladně zodpovězený dotaz z obsahuje y certifikát w dokládají příslušnost dotazovaného řetězce do orákula S.

Zbývá ukázat, že $R \in \Pi_{i-1}^P$. Nejprve je nutné ověřit, že všechny kroky stroje $M^?$ jsou přípustné – to lze provést v polynomiálním čase. Dále je třeba zkontrolovat pro polynomiálně mnoho dvojic (z_i, w_i) zda $(z_i, w_i) \in S$. Ale $S \in \Pi_{i-2}^P$, tím spíše tedy $S \in \Pi_{i-1}^P$. Pro všechny záporně zodpovězené orákulové dotazy z_i' je ještě třeba zkontrolovat, že $z_i' \notin K$. To je ovšem dotaz z třídy Π_{i-1}^P.

Celkem tedy je tedy možné rozhodnout, zda $(x, y) \in R$ v polynomiálním čase s použitím dotazů na orákulum z Π_{i-1}^P. □

Věta 2.11. *Jazyk $L \in \Pi_i^{\mathsf{P}}$, právě když existuje polynomiálně vyvážená relace R tak, že*

$$R \in \Sigma_{i-1}^{\mathsf{P}},$$
$$L = \{x \mid \forall y.|y| \leq |x|^k.(x,y) \in R\}.$$

Důkaz. Analogicky. □

Důsledek 2.12. *$L \in \Sigma_i^{\mathsf{P}}$, právě když existuje polynomiálně vyvážená a ověřitelná $(i+1)$-ární relace R taková, že*

$$L = \{x \mid \exists y_1 \forall y_2 \exists y_3 \ldots Q y_i.(x, y_1, y_2, \ldots, y_i) \in R\}.$$

Věta 2.13. $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PH} \subseteq \mathsf{PSPACE}$.

Věta 2.14. *Pokud $\Sigma_i^{\mathsf{P}} = \Pi_i^{\mathsf{P}}$ pro nějaké i, potom $\forall j > i.\Sigma_j^{\mathsf{P}} = \Sigma_i^{\mathsf{P}}$.*

Důkaz. Stačí ukázat, že $\Sigma_i^P = \Pi_i^P \Rightarrow \Sigma_{i+1}^P = \Sigma_i^P$. Uvažme tedy jazyk $L \in \Sigma_{i+1}^P$. Podle věty 2.11 existuje relace $R \in \Pi_i^P$ taková, že $L = \{x \mid \exists y.(x,y) \in R\}$. Protože však $\Sigma_i^{\mathsf{P}} = \Pi_i^{\mathsf{P}}$, platí $R \in \Sigma_i^P$, nebo-li $(x,y) \in R \iff \exists z.(x,y,z) \in S$ pro nějaké $S \in \Pi_{i-1}^P$.

Celkem pak $x \in L$ právě tehdy, když existuje řetězec $y;z$ takový, že $(x,y,z) \in S$, kde $S \in \Pi_{i-1}^P$. Ale to znamená, že $L \in \Sigma_i^P$. □

Důsledek 2.15. *Pokud $\mathsf{P} = \mathsf{NP}$ (resp. $\mathsf{NP} = \mathsf{coNP}$), polynomiální hierarchie kolabuje na nulté (resp. první) úrovni.*

2.2 Alternující Turingovy stroje

Poznámka. Připomeňme, že výpočtem nedeterministického Turingova stroje je strom, jehož uzly jsou ohodnoceny stavy stroje. Navíc každý list výpočtu je ohodnocen buď akceptujícím, nebo zamítajícím stavem. Stroj akceptuje slovo x, právě když alespoň jeden list výpočtu je akceptující.

Definice 2.16 (Alternující Turingův stroj)**.** Jediná změna oproti Turingovým strojům je množina stavů $Q = Q_\exists \cup Q_\forall$, přičemž množiny $Q_\exists$ a $Q_\forall$ jsou disjunktní. Výpočet ATS na slově w je konečný strom, stejně jako u NTS. Každý uzel ve stromu je buď existenční nebo univerzální.

Listy jsou akceptující nebo zamítající. Výpočetní podstrom s univerzálním kořenem akceptuje, právě když všichni následníci kořene akceptují. Výpočetní podstrom s existenčním kořenem akceptuje, právě když alespoň jeden následník kořene akceptuje. ATS akceptuje výpočet právě když jeho výpočet je akceptující.

Poznámka. Složitostní míry: čas (hloubka výpočetního stromu), prostor (velikost použité pásky), počet alternací (maximum přes všechny cesty). Můžeme definovat složitostní třídy $\mathsf{ATIME}(t(n)), \mathsf{ASPACE}(s(n)), \ldots$

Asi lze mluvit i o konečných automatech, jak se zdá z následujícího příkladu.

Příklad. Dvouhlavý alternující konečný automat (2-AFA) rozhodující jazyk $L = \{a^n \mid n = 2^i\}$.

1. H1 se zůstane na místě, H2 se posune na střed slova (nedeterministicky ho uhodnu).
2. Univerzálně rozdělíme výpočet na dva.
   (a) Verifikuje, že H2 je skutečně ve středu řetězce. H1 se bude posunovat od dvě pole, H2 o jedno.
   (b) Posuneme H1 na pozici H2 (opět nedeterministicky), univerzálně rozdělíme na dva výpočty. První verifikuje, že H1 a H2 jsou na stejné pozici, druhý provede celý výpočet znovu rekurzivně.

Stroj bude mít lineární časovou složitost. (Může mít vícehlavový AFA jinou složitost? Jasně, že ne.)

Věta 2.17. *Pro $f(n) \geq n$ platí*

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

Pro $f(n) \geq \log n$ platí

$$\mathsf{ASPACE}(f(n)) = \mathsf{DTIME}(2^{O(f(n))}) \subseteq \mathsf{ASPACE}(O(f(n))).$$

Hierarchie tedy vypadá takto:

$$\mathsf{DLOG} \subseteq \mathsf{P} = \mathsf{ALOG} \subseteq \mathsf{AP} = \mathsf{PSPACE} \subseteq \mathsf{APSPACE} = \mathsf{DEXPTIME}.$$

Důkaz.

- $\mathsf{ATIME}(f(n)) \subseteq \mathsf{DSPACE}(f(n))$: Simulací. DTS S simuluje ATS M prohledáváním výpočetního stroju M do hloubky. S akceptuje, pokud kořen stromu je akceptující.

  S potřebuje k uložení informací pro DFS zásobník. Jeho hloubka je $f(n)$. Každá konfigurace může mít velikost až $f(n)$ (horní odhad prostorové složitosti M je jeho časová složitost. Pak tedy S vyžaduje $f^2(n)$ paměti.

Povšimněme si dále, že není třeba ukládat celé konfigurace. Postačí uložit pouze nedeterministické volby učiněné v jednotlivých uzlech. Danou konfiguraci je možno v případě potřeby znovu vypočítat simulací M řízenou posloupností nedeterministických voleb. Pak stačí prostor velikosti $f(n)$, což bylo dokázat.

- $\mathsf{DSPACE}(f(n)) \subseteq \mathsf{ATIME}(f^2(n))$: Modifikací Savitchovy věty. Nechť M je DTS, $M \in \mathsf{DSPACE}(f(n))$. Zkonstruujeme ATS $S \in \mathsf{ATIME}(f^2(n))$ simulující M.

  Uvažme konfigurace stroje M c_1, c_2 a číslo t. Potřebujeme otestovat, zda M může přejít z c_1 do c_2 provedením t kroků. Alternující procedura pro tento problém bude pracovat následujícím způsobem:

  - Rozdělí se existenčně a uhádne c_m na polovině cesty z c_1 do c_2.
  - Rozdělí se univerzálně a rekurzivně testuje, zda je možné dostat se z c_1 do c_m a z c_m do c_2 prostřednictvím $t/2$ kroků.

  S použije tuto proceduru ke zjištění, zda je z počáteční konfigurace dosažitelná akceptující konfigurace v $2^{df(n)}$, kde d je vybráno tak, že M nemá více než $2^{df(n)}$ konfigurací.

  Libovolná větev potřebuje $O(f(n))$ času na zápis konfigurace v jednom uzlu. Hloubka rekurze je $\log 2^{df(n)} = O(f(n))$. Celkem tedy $S \in \mathsf{ATIME}(f^2(n))$.

- $\mathsf{ASPACE}(f(n)) \subseteq \mathsf{DTIME}(2^{O(f(n))})$: Nechť $M \in \mathsf{ASPACE}(f(n))$. DTS S bude prohledávat graf konfigurací M na slově w. List označí za akceptující, pokud obsahuje akceptující konfiguraci. Univerzální list označí za akceptující, pokud všichni jeho následníci jsou akceptující a existenční list označí za akceptující, pokud alespoň jeden jeho následník je akceptující. S akceptuje, je-li po průchodu stromem označen kořen.

  Počet konfigurací M je $2^{O(f(n))}$, neboť $f(n) \geq \log n$. Proto je velikost grafu $2^{O(f(n))}$. Konstrukce grafu i jeho prohledání lze provést přibližně v takovém čase. Proto je celkový potřebný čas $2^{O(f(n))}$, a tedy $D \in \mathsf{DTIME}(2^{O(f(n))})$.

- $\mathsf{DTIME}(2^{O(f(n))}) \subseteq \mathsf{ASPACE}(f(n))$: Ukážeme, jak simulovat DTS $M \in \mathsf{DTIME}(2^{O(f(n))})$ strojem $S \in \mathsf{ASPACE}(f(n))$. Simulace není snadná, neboť S má k dispozici velice málo prostoru.

  Základem simulace je tabulka konfigurací podobná tabulce použité v důkazu věty 1.7 s tím rozdílem, že buňka může obsahovat jak symboly

abecedy, tak označení stavu. Kvůli nedostatku prostoru si S nebude pamatovat celou tabulku, nýbrž jen indexy do tabulky.

Uvažme následující situaci někde v tabulce:

a	b	c
	d	

Obsah buňky d lze určit z obsahu buněk a, b a c. S rekurzivně hádá a následně ověřuje obsah jednotlivých buňek tabulky. K ověření obsahu buňky d, která není v prvním řádku, simulátor S nejprve existenčně uhádne obsah rodičovských buňek (a, b, c) a následně se univerzálně rozvětví k ověření správnosti těchto hádání. Ověřuje rekurzivně. Je-li d na prvním řádku, S dokáže rovnou odpovědět, neboť zná počáteční konfiguraci M. Můžeme předpokládat, že M při akceptování přesune hlavu na nejlevější políčko, takže S může zjistit, zda M akceptuje, kontrolou odpovídající buňky v tabulce. S si potřebuje pamatovat pouze jeden index do tabulky; na to je třeba $\log 2^{O(f(n))} = O(f(n))$ prostoru.

□

Definice 2.18. Složitostní třída Σ_i-$\mathsf{ATIME}(f(n))$ – max i alternací, první je existenční. Složitostní třída Π_i-$\mathsf{ATIME}(f(n))$ – max i alternací, první je univerzální.

Lema 2.19.
$\Sigma_i^{\mathsf{P}} = \Sigma_i\text{-}\mathsf{ATIME}(n^{O(1)}), \Pi_i^{\mathsf{P}} = \Pi_i\text{-}\mathsf{ATIME}(n^{O(1)}), \mathsf{AP} \supseteq \bigcup_i \Sigma_i\text{-}\mathsf{ATIME}(n^{O(1)})$.

Poznámka. ATS vlastně reprezentují paralelismus. Existují také synchronizované ATS (SATS). Synchronizace opět posune třídy o jeden krok.

2.3 Paralelní výpočetní modely

Poznámka. Paralelních modelů je mnoho, my budeme pracovat s logickými obvody.

Definice 2.20 (Běžné logické obvody)**.** Vždy pracujeme s rodinou (posloupností) obvodů C_n, kde n je počet vstupů. Velikostí obvodu rozumíme počet hradel, hloubkou rozumíme délku nejdelší cestu ze vstupu do výstupu hradla.

Definice 2.21. Posloupnost obvodů je uniformní, pokud existuje algoritmus, který z n vypočítá C_n. Analogicky definujeme $\mathsf{LOGSPACE}$ uniformitu, polyuniformitu atd.

Poznámka. Umíme zadefinovat třídu $\mathsf{DEPTHSIZE}(f, g)$, kde $f(n)$ je velikost obvodu C_n a $g(n)$ je hloubka obvodu C_n. Analogií klasické třídy P je v případě logických obvodů třída obvodů s polynomiální velikostí a logaritmickou hloubkou.

Definice 2.22. NC^i je třída jazyků pro které existuje $\mathsf{LOGSPACE}$-uniformní posloupnost logických obvodů polynomiální velikosti a hloubky $O(\log^i n)$. Dále definujeme

$$\mathsf{NC} = \bigcup_{i=0}^{\infty} \mathsf{NC}^i$$

Poznámka. Třída NC je pojmenovaní po Nicku Pippengerovi (Nick's Class).

Věta 2.23.

$$\mathsf{NC}^1 \subseteq \mathsf{DLOG},$$
$$\mathsf{NLOG} \subseteq \mathsf{NC}^2,$$
$$\mathsf{NC} \subseteq \mathsf{P}.$$

Důkaz.

- $\mathsf{NC}^1 \subseteq \mathsf{DLOG}$: sestavíme log-space algoritmus rozhodující jazyk $A \in \mathsf{NC}^1$. Pro slovo w délky n je možné zkonstruovat $n$tý logický obvod pro A. Algoritmus pak může vyhodnotit obvod užitím DFS z výstupního uzlu. Je třeba pamatovat si pouze cestu do aktuálního uzlu a výsledky vypočítané během této cesty. Protože obvod má logaritmickou hloubku, je třeba jen logaritmický prostor.

- $\mathsf{NLOG} \subseteq \mathsf{NC}^2$: Buď A jazyk (bez újmy na obecnosti nad binární abecedou) akceptovaný strojem $M \in \mathsf{NL}$. Zkonstruujeme posloupnost logických obvodů $C_0, C_1, \ldots$ pro A. C_n vytvoříme jako graf G podobný výpočetnímu grafu pro M na vstupu w délky n. Známe pouze n, nikoliv w. Vstupy obvodu jsou proměnné $w_1, \ldots, w_n$ odpovídající znakům w.

  Konfigurace stroje M nad vstupem w popisuje stav, obsah pracovní pásky a pozice vstupní a pracovní hlavy. Povšimněme si, že neobsahuje samotné slovo w. Proto množina konfigurací ve skutečnosti závisí jen na n. Protože $M \in \mathsf{NLOG}$, existuje jen polynomiální počet konfigurací. Ty tvoří uzly grafu G.

  Hrany G jsou označeny vstupními proměnnými w_i. Jsou-li c_1, c_2 uzly G a c_1 má vstupní hlavu na pozici i, přidáme do G hranu (c_1, c_2) s popisem w_i (nebo $\overline{w_i}$), pokud lze z c_1 přejít do c_2 v jednom kroku, přičemž $w_i = 1$

(nebo $w_i = 0$). Pokud lze přechod provést nezávisle na w_i, zůstává hrana bez popisu.

Nastavíme-li hrany podle řetězce w délky n, existuje cesta z počáteční konfigurace do akceptující konfigurace právě když M akceptuje w. Obvod tedy počítá tranzitivní uzávěr G a vrací přítomnost cesty, která akceptuje všechny řetězce z A délky n. Obvod má polynomiální velikost a hloubku $O(\log^2 n)$.

- $\mathsf{NC} \subseteq \mathsf{P}$: Algoritmus s polynomiální časovou složitostí je schopen spustit log-space překladač k výpočtu C_n a následně simulovat výpočet obvodu na vstupu délky n.

□

Poznámka. Poslední inkluze není překvapující a znamená, že paralelismus nám nepřinese nic navíc.

Neví se, zda jsou inkluze ostré.

Lema 2.24. *Pokud* P*-úplný problém patří do* NC*, pak* $\mathsf{NC} = \mathsf{P}$.

Poznámka. Příkladem P-úplných problémů jsou

- Circuit Value Problem,
- DFS (pro graf s fixovaným pořadím následníků a kořenem, rozhodni pro dva vrcholy, který bude navštívený dřív),
- BFS (analogicky).

Příkladem problému, který umíme paralelně počítat rychleji než sekvenčně, je násobení matic.

2.4 Pravděpodobnostní výpočty

Poznámka. Opět chceme najít model, definovat kritéria složitosti, definovat třídu praktických problémů a zařadit ji do hierarchie.

Definice 2.25. Pravděpodobnostní Turnigův stroj je DTS se speciální read-only páskou, na které je zapsaný nekonečně dlouhý řetězec (random string). Stroj se po random pásce může posouvat pouze doprava.

Poznámka. Budeme pracovat s časově ohraničenými PTS, není tedy nutné počítat s nekonečnou random páskou.

Definice 2.26. Pravděpodobnost, že PTS M akceptuje vstup x definujeme jako

$$\Pr_y(M(x,y) \text{ akceptuje}) = \frac{|\{y \in \{0,1\}^k \mid M(x,y) \text{ akceptuje}\}|}{2^k},$$

kde $k \geq T(n)$.

Poznámka. Složitostní míry zůstávají jako u deterministických výpočtů: čas a prostor. Navíc přibývá délka náhodného řetězce.

Definice 2.27. Jazyk A patří do RP, když existuje PTS M s polynomiální časovou složitostí n^c takový, že

- pokud $x \in A$, $\Pr_y(M(x,y) \text{ akceptuje}) \geq \frac{3}{4}$,
- pokud $x \notin A$, $\Pr_y(M(x,y) \text{ akceptuje}) = 0$.

Jazyk A patří do BPP, když existuje PTS M s polynomiální časovou složitostí n^c takový, že

- pokud $x \in A$, $\Pr_y(M(x,y) \text{ akceptuje}) \geq \frac{3}{4}$,
- pokud $x \notin A$, $\Pr_y(M(x,y) \text{ akceptuje}) \leq \frac{1}{4}$.

Věta 2.28. $\mathsf{P} \subseteq \mathsf{RP} \subseteq \mathsf{BPP} \subseteq \mathsf{PSPACE}$, $\mathsf{RP} \subseteq \mathsf{NP}$

Poznámka. Všiměme si, že nedeterminismus je zvláštní případ náhodnosti. Na rozdíl od RP však k akceptování stačí jeden akceptující výpočet.

Vztah BPP a NP je zcela otevřený. Všeobecně převládající názor je, že $\mathsf{BPP} \subseteq \mathsf{P}$.

Definice 2.29. Definujeme odvozenou třídu $\mathsf{ZPP} = \mathsf{RP} \cap \mathsf{coRP}$.

Poznámka. Pro jazyky v ZPP můžeme sestrojit Turingův stroj, který v polynomiálním čase vrací odpověď ano/ne/nevím, přitom odpověď nevím je pravděpodobnostně ohraničená.

Poznámka. Zásadní vlastnost pravděpodobnostních algoritmů, bez které by neměli smysl je amplifikace.

Lema 2.30. *Pokud $A \in \mathsf{RP}$, pro každý polynom n^d existuje PTS s polynomiální časovou složitostí takový, že pro každý vstup x délky n platí*

- *pokud $x \in A$, $\Pr_y(M(x,y)$ akceptuje$) \geq 1 - 2^{-n^d}$,*
- *pokud $x \notin A$, $\Pr_y(M(x,y)$ akceptuje$) = 0$.*

Pokud $A \in$ BPP, pro každý polynom n^d existuje PTS s polynomiální časovou složitostí takový, že pro každý vstup x délky n platí

- *pokud $x \in A$,* $\Pr_y(M(x,y)$ *akceptuje*$) \geq 1 - 2^{-n^d}$,
- *pokud $x \notin A$,* $\Pr_y(M(x,y)$ *akceptuje*$) \leq 2^{-n^d}$.

Důkaz.

- RP: Buď M PTS s polynomiální časovou složitostí takový, že
  - Je-li $x \in A$, pak $\Pr_y(M(x,y)$ akceptuje$) \geq \frac{3}{4}$
  - Je-li $x \notin A$, pak $\Pr_y(M(x,y)$ akceptuje$) = 0$

  Zkonstruujeme další PTS N, který pro vstup x spustí M n^dkrát za použití čerstvých náhodných bitů pro každý běh. N akceptuje, pokud M v některém běhu akceptuje. N spustí polynomiální počet polynomiálních výpočtů, takže jeho časová složitost je rovněž polynomiální.

  Pokud $x \notin A$, M pokaždé zamítne, a tedy i N zamítne. Pokud $x \in A$, je pravděpodobnost chybné odpovědi N

$$\begin{aligned}\Pr_{y_1,\ldots,y_{n^d}}(N(x,y_1,\ldots,y_{n^d}) \text{ zamítá}) &= \prod_{i=1}^{n^d} \Pr_{y_i}(M(x,y_i) \text{ zamítá}) \\ &= \Pr_y(M(x,y) \text{ zamítá })^{n^d} \\ &\leq 4^{-n^d}.\end{aligned}$$

- BPP: Konstrukce je analogická, mění se pouze podmínka akceptování. N provede n^{d+1} výpočtů a jako svůj výstup vrátí převládající výsledek. Pravděpodobnost chyby je rovna pravděpodobnosti, že alespoň polovina z n^{d+1} výsledků je správná. Je možné dopočítat, že tato pravděpodobnost je $\leq 2^{-n^d}$ pro dostatečně velké n.

□

Poznámka. Buď $m = n^c$ maximální počet bitů čtených z random pásky stroje M. Definujeme dvě pomocné třídy:

$$\begin{aligned}A_x &= \{y \in \{0,1\}^m \mid M(x,y) \text{ akceptuje}\}, \\ R_x &= \{y \in \{0,1\}^m \mid M(x,y) \text{ zamítá}\}.\end{aligned}$$

Pokud $x \in A$, $|A_x| \geq 2^m - 2^{m-n}$, $|R_x| \leq 2^{m-n}$. Naopak, pokud $x \notin A$, $|A_x| \leq 2^{m-n}$, $|R_x| \geq 2^m - 2^{m-n}$.

Lema 2.31. *$x \in A$ právě když existuje m retězců $z_1, \ldots, z_m$ takových, že $\{y \oplus z_j \mid 1 \leq j \leq m, y \in A_x\} = \{0,1\}^m$.*

Důkaz. Buď $x \notin A$. Potom $|A_x| \leq 2^{m-n}$. Pro libovolné $z_1, \ldots, z_m$ dostáváme

$$\{y \oplus z_j \mid 1 \leq j \leq m, y \in A_x\} = \bigcup_{j=1}^{m} \{y \oplus z_j \mid y \in A_x\}$$

Horní odhad velikosti této množiny je

$$\sum_{j=1}^{m} |\{y \oplus z_j \mid y \in A_x\}| = \sum_{j=1}^{m} |A_x| \leq m \cdot 2^{m-n} \leq 2^m$$

Naopak pokud $x \in A$, pak $|R_x| \leq 2^{m-n}$. Nazývejme prvky $z_1, \ldots, z_m$ *špatné*, pokud pro nějaké w platí

$$\{w \oplus z_j \mid 1 \leq j \leq m\} \subseteq R_x$$

a *dobré* jinak. Chceme ukázat, že existují dobré $z_1, \ldots, z_m$. Ovšem všechny špatné $z_1, \ldots, z_m$ jsou určeny podmnožinou R_x velikosti m, kterých je nejvýše $(2^{m-n})^m$ a řetězcem $w \in \{0,1\}^m$, kterých je 2^m. Horní odhad na počet špatných $z_1, \ldots, z_m$ je

$$(2^{m-n})^m \cdot 2^m = 2^{m(m-n+1)} < 2^{m^2}$$

a pravá strana je celkový počet možných $z_1, \ldots, z_m$. Proto nějaké $z_1, \ldots, z_m$ musí být dobré. □

Věta 2.32. $\mathsf{BPP} \subseteq \Sigma_2 \cap \Pi_2$

Důkaz. Dokážeme $A \in \mathsf{BPP} \Rightarrow A \in \Sigma_2$.

1. Uhodni $z_1, \ldots, z_m$ (existenčně).
2. Vygeneruj všechny w délky m (univerzálně).
3. Ověř, že $w \in \{y \oplus z_j \mid 1 \leq j \leq m, y \in A_x\}$ nebo ekvivalentně $\{w \oplus z_j | mid 1 \leq j \leq m\} \cap A_x \neq \emptyset$.

Třetí bod lze provést deterministicky, takže konstrukce obsahuje dvě úrovně hádání, přičemž nezáleží na jejich pořadí. Celkem se tedy jedná o Σ_2^P i Π_2^P výpočet. □

2.5 Interaktivní protokoly

Máme dva subjekty (P, proover, a V, verifier), oba vidí vstupní řetězec, mohou spolu spolupracovat a nakonec jeden z nich vrátí odpověď. Stroj V je pravděpodobnostní Turingův stroj bežící v polynomiálním čase, na stroj P nejsou kladená žádná omezení.

Model byl vymyšlený třemi nezávislými skupinami během jednoho roku, motivace se mírně lišily, ale šlo o kryptografii.

- Golwasser, Micali, Rachoff,
- Babai – Arthur-Merlin games,
- Papadimitriau – Games Against Nature.

Definice 2.33. Ověřovatel je funkce $V : (\Sigma^*)^3 \to \Sigma^* \cup \{\text{accept}, \text{reject}\}$ se třemi vstupy: vstupní řetězec, náhodný řetězec, komunikační historie. Komunikace je řetězec $m_1 \# m_2 \# \ldots \# m_i$. Výstupem je zpráva odesílaná dokazovateli. Dokazovatel je funkce P. Vstupní řetězec, komunikační historie. Výstupem je zpráva pro ověřovatele. Pro celý model píšeme $(P \leftrightarrow V)(w, r) =$ accept/reject, mluvíme o pravděpodobnosti.

Definice 2.34 (Třída IP)**.** Jazyk $A \in \mathsf{IP}$, právě když existuje polynomiálně časově vypočítatelná funkce V a (neohraničená) funkce P tak, že pro každou funkci $\widetilde{P}$ a řetězec w platí

- pokud $w \in A$, $\Pr_r[(V \leftrightarrow P)(w, r)$ akceptuje$] \geq \frac{2}{3}$,
- pokud $w \in A$, $\Pr_r[(V \leftrightarrow \widetilde{P})(w, r)$ akceptuje$] \leq \frac{1}{3}$.

Poznámka. Třída IP reprezentuje problémy „prakticky“ řešitelné interaktivními protokoly. Třída je amplifikovatelná.

Cvičení. Bez pravděpodobnosti by celý systém nebyl silnější než NP. (Proč?)

Příklad. Problém neisomorfismu grafů $\{(G_1, G_2) \mid G_1 \cong G_2\}$.

1. Ověřovatel náhodně zvolí $j = 1, 2$. Zvolíte náhodnou permutaci vrcholů Π. Odešle $\Pi(G_j)$.
2. Pokud jsou grafy neizomorfní, dokazovatel si spočítá, které j si ověřovatel zvolil, jinak si j zvolí náhodně. j pak odešle ověřovateli.
3. Ověřovatel akceptuje, právě když dokazovatel odeslal správné j.

Lema 2.35. *Problém*

$$\#SAT = \{< \varphi, k > |\ \varphi \text{ je v 3CNF, počet splňujících přiřazení } \varphi \text{ je } k\}$$

patří do IP.

Důkaz. Zkonstruujeme nteraktivní protokol pro #SAT. Definujeme rodinu funkcí $f_i(a_1, \ldots, a_i) =$ počet splňujících přiřazení formule $\varphi | x_1 = a_1, \ldots, x_i = a_i$. Tedy $f_0()$ je počet splňujících přiřazení formule φ. $f_n(a_1, \ldots, a_n) = 1$, právě když $\varphi(a_1, \ldots, a_n)$ je pravdivá. Dále platí

$$f_i(a_1, \ldots, a_i) = f_{i+1}(a_1, \ldots, a_i, 0) + f_{i+1}(a_1, \ldots, a_i, 1).$$

Verifier nyní požádá provera o $f_0()$ a zkontroluje, že výsledek je k. Potom požádá o $f_1(0)$ a $f_1(1)$ a zkontroluje, že $f_0() = f_1(0) + f_1(1)$. A tak dále. Nakonec pro každé $(a_1, \ldots, a_n)$ zkontroluje, že $f_n(a_1, \ldots, a_n) = 1$, právě když $\varphi(a_1, \ldots, a_n)$ platí.

Tento algoritmus má exponenciální složitost.

Funkci tedy rozšíříme na $f_i : \mathbb{R}^i \to \mathbb{R}$. Definujeme

$$f_i(a_1, \ldots, a_i) = \sum F(a_1, \ldots, a_n)$$

, kde polynom F je aritmetizací formule φ definovaný induktivně:

- $\alpha \wedge \beta \to \alpha \cdot \beta$,
- $\neg\alpha \to 1 - \alpha$,
- $\alpha \vee \beta \to 1 - (1 - \alpha)(1 - \beta)$.

Označme operaci odpovídající $\vee$ symbolem $*$. Podmínky popsané v předchozím pokusu stále platí. Stupeň polynomu je maximálně $3l$, kde l je délka formule.

1. Verifier požádá provera o $f_0()$, nějaké dostatečně velké prvočíslo $q < 2^n$ a certifikát prvočíselnosti q. Verifier ověří prvočíselnost q a $f_0() = k$. Všechny další komunikace jsou modulo q.
2. Prover pošle polynom $\lambda z.f_1(z)$, ověřovatel zkontroluje $f_1(0) + f_1(1) = f_0()$, stejně jako v minulém pokusu.
3. Verifier zvolí náhodně celé číslo $0 \leq r_1 < q$ a pošle ho dokazovateli. Dokazovatel pošle $\lambda z.f_2(r_1, z)$. Ověřovatel zkontroluje.
4. V každém dalším kroku ověřovatel pošle $(r_1, \ldots, r_i)$, dokazovatel pošle $\lambda z.f_{i+1}(r_1, \ldots, r_i, z)$. Ověřovatel zkontroluje.

5. Nakonec se zkontroluje, zda $p(r_1, \ldots, r_n) = f_n(r_1, \ldots, r_n)$.

□

Věta 2.36. IP = PSPACE.

Důkaz. Důkaz se rozpadá do dvou částí.

1. IP ⊆ PSPACE. Na této inkluzi není nic zajímavého. Máme jazyk $L \in$ IP, máme tedy konkrétního dokazovatele a ověřovatele. Deterministicky simulujme výpočet přes všechny možné dokazovatele a všechny náhodné řetězce a spočítáme pravděpodobnost správného výsledku. To uděláme v polynomiálním prostoru.

2. IP ⊇ PSPACE. Najdeme interaktivní protokol pro nějaký PSPACE-úplný problém, v našem případě pro QBF (pravdivost kvantifikovaných formulí).

   $\psi = Q_1 x_q Q_2 x_2 \ldots Q_n x_n[\varphi]$.

   $$f_i(a_1, \ldots, a_i) = \begin{cases} 1 & \text{pokud } Q_{i+1}x_{i+1} \ldots Q_n x_n \varphi[x_1 = a_1 \ldots x_i = a_i] \\ & \text{je splnitelná} \\ 0 & jinak \end{cases}$$

   Po aritmetizaci má polynom příliš velký stupeň, je tedy potřeba následující trik.

   φ přepíšeme na $\psi = Qx_1 Rx_1 Qx_2 Rx_1 Rx_2 Qx_3 Rx_1 Rx_2 Rx_3 \ldots [\varphi]$. Označíme $\psi' = S_1 y_1 S_2 y_2 \ldots S_k y_k[\varphi]$, kde $S_i \in \{\forall, \exists, R\}$.

   Pro $i \leq k$ definujeme funkci f_i. Nechť $f_k(x_1, \ldots, x_n)$ je definována jako polynom $p(x_1, \ldots, x_m)$ vzniklý aritmetizací formule φ. Pro $i < k$ pak f_i definujeme takto:

   - $S_i = \forall$, pak $f_i(\ldots) = f_{i+1}(\ldots, 0) \cdot f_{i+1}(\ldots, 1)$.
   - $S_i = \exists$, pak $f_i(\ldots) = f_{i+1}(\ldots, 0) * f_{i+1}(\ldots, 1)$, kde $x * y = 1 - (1 - x)(1 - y)$.
   - $S_i = R$, pak $f_i(\ldots, a) = (1 - a) f_{i+1}(\ldots, 0) + a f_{i+1}(\ldots, 1)$.

   Povšimněme si, že operace Rx nemění hodnotu polynomu na boolovských hodnotách, ale výsledný polynom je lineární v x. V ψ' jsme po $Q_i x_i$ přidali $R_{x_1} \ldots R_{x_i}$, abychom snížili stupeň všech proměnných. Nyní můžeme přistoupit k popisu protokolu.

- *Fáze 0*: Prover pošle $f_0()$. Verifier zkontroluje, zda $f_0 = 1$ a zamítne, pokud tomu tak není.
- *Fáze i*: Podobně jako v protokolu pro SAT se prover snaží dokázat $f_{i-1}(r_1, \dots)$ předložením $f_i(r_1, \dots, r)$.

  Prover zašle koeficienty $f_i(r_1, \dots, z)$ jako polynom v proměnné z. Verifier použije tyto koeficienty k výpočtu hodnot $f_i(r_1, \dots, 0)$ a $f_i(r_1, \dots, 1)$ a zkontroluje

  $$\begin{aligned} f_{i-1}(r_1, \dots) &= f_i(r_1, \dots, 0) \cdot f_i(r_1, \dots, 1) \text{ pro } S = \forall \\ f_{i-1}(r_1, \dots) &= f_i(r_1, \dots, 0) * f_i(r_1, \dots, 1) \text{ pro } S = \exists \\ f_{i-1}(r_1, \dots, r) &= (1-r) f_i(r_1, \dots, 0) + r f_i(r_1, \dots, 1) \text{ pro } S = R \end{aligned}$$

  Pokud některá z rovností neplatí, verifier odmítá.

  Verifier pošle náhodné číslo r. Pokud $S = R$, toto r nahrazuje předešlé r.
- *Fáze $k+1$*: Verifier přímo ověří, zda $p(r_1, \dots, r_m) = f_m(r_1, \dots, r_m)$. Akceptuje právě když jsou obě hodnoty stejné.

Důkaz korektnosti protokolu je podobný důkazu pro #SAT. Je-li ψ pravdivá, může prover postupovat podle protokolu a verifier nakonec akceptuje. Pokud ψ pravdivá není, musí zlý prover lhát v kroku 0. V kroku i pak musí vždy alespoň jednu z hodnot poslat nesprávnou.

Pravděpodobnost, že by zlý prover měl štěstí pro náhodné r je nízká (stupeň polynomu podělený velikostí prvočísla). Nutno podotknout, že proverovi stačí úspěch v jediné fázi, to mu umožní pokračovat s již pravdivými hodnotami.

□

Poznámka. Rovnost tříd vyvolala rozruch v komunitě.

3 Technika přechodových posloupností

Předpokládáme, že akceptující konfigurace má hlavu úplně vpravo.

Pracujeme s přechodovými posloupnostmi: sledujeme posloupnost stavů, ve kterých se stroj během výpočtu nachází, když jeho hlava překonává hranici mezi polem i a $i+1$ na vstupní pásce. Takovou posloupnost stavů značíme $i/i+1$.

Součet délek přechodových posloupností dává odhad na časovou složitost.

Lema 3.1. *Nechť jednopáskový DTS T akceptuje L a nechť V je výpočet T na $w = a_1, \ldots, a_n$ a nechť přechodová posloupnost mezi p_i a p_{i+1} a mezi p_j a p_{j+1} jsou stejné ($i < j$). Potom T akceptuje slovo $a_1, \ldots, a_i, a_{j+1}, \ldots, a_n$.*

Dvoupáskový TS akceptující dvojici slov $a_1 \ldots a_n$, $b_1 \ldots b_m$ akceptuje i dvojici $a_1 \ldots a_i b_{j+1} \ldots b_m$, $b_1 \ldots b_j a_{i+1} \ldots a_n$, jestliže přechodové posloupnosti $i/i+1$ na první pásce a $j/j+1$ na druhé jsou stejné.

Věta 3.2. *Jazyk $L = \{w\#w^R \mid w \in \{a,b\}^*\}$. Pokud T rozhoduje L, pak* $\mathsf{TIME}_T(n) = \Omega(n^2)$.

Důkaz. $L_n = \{w\#w^R \mid |w| = n\}$. Fixujeme i, $p(i)$ bude průměrná délka přechodové posloupnosti mezi p_i a p_{i+1} pro slova z L_n. Počet různých přechodových posloupností délky $2p(i)$ je zhora ohraničený $s^{2p(i)+1}$, kde s je počet stavů. Alespoň polovina slov z L_n má p.p. na i délky $\leq 2p(i)$. (Proč?) Z toho: existuje aspoň

$$A = \frac{\frac{1}{2}2^n}{s^{2p(i)+1}}$$

slov z L_n, které mají přiřazené stejné p.p.

Počet slov z L_n, které se liší na pozicích $i+1$ až n – takových je

$$B = 2^{n-i}.$$

Pokud $A > B$, pak exustují dvě slova $w_1 w_2$ a $w_3 w_2$ se shodnou p.p. a shodující se na $i+1, \ldots, n$, tedy $w_1 \neq w_3$. Potom ale stroj akceptuje i slovo $w_1 w_2 \# w_2^R w_3^R$.

Proto pokud stroj rozhoduje L, musí platit $A \leq B$. Lze odvodit $p(i) \geq \frac{i-1}{3 \log s}$.

Potom $\sum_{i=1}^{n} p(i) \geq \frac{n^2-4n+3}{24 \log s}$. Proto $T(n) \geq \sum_{i=1}^{n} p(i) \in \Omega(n^2)$. □