KMI/SLO - poznámky

Verze 28/11/2023 v 15:07:37

Petr Osička

Disclaimer: učební text pro kurz složitosti,

Obsah

1 Základní složitostní třídy a jejich vlastnosti

Nejdříve definujeme časovou a prostorovou složitost TS (přitom se omezíme na TS, které vždy zastaví).

- TS M běží v čase $T(n)$, pokud pro všechny vstupy x platí, že M pro x provede méně než $T(|x|)$ kroků.
- TS M běží s pamětí $S(n)$, pokud všechny x platí, že M během výpočtu se vstupem x navštíví nejvýše $S(|x|)$ různých políček pásky.
- Pro NTS jsou definice analogické, pouze bereme v úvahu všechny výpočetní větve (tedy omezení pomocí $T(n)$ a $S(n)$ platí pro každou výpočetní větev)

Dále pak zavedeme základní složitostní třídy, které jim odpovídají.

$$\begin{aligned}
\mathbf{DTIME}(T(n)) &= \{L(M) \mid M \text{ je TS běžící v čase z } O(T(n))\} \\
\mathbf{NTIME}(T(n)) &= \{L(M) \mid M \text{ je NTS běžící v čase z } O(T(n))\} \\
\mathbf{DSPACE}(S(n)) &= \{L(M) \mid M \text{ je TS běžící s pamětí z } O(S(n))\} \\
\mathbf{NSPACE}(S(n)) &= \{L(M) \mid M \text{ je NTS běžící s pamětí z } O(S(n))\}
\end{aligned}$$

V definicích je skryt tzv. *linear speedup theorem*, který říká, že k TS běžícímu v čase $cT(n)$ (kde c je konstanta) existuje TS rozhodující stejný jazyk běžící v čase $T(n)$. Věta se dokazuje pomocí komprese páskové abecedy, kdy c políček pásky reprezentujeme jedním a c kroků prvního TS pak odpovídá jednomu kroku druhého TS (dostaneme tak TS s větší páskovou abecedou). Analogická věta platí i pro pamětovou složitost a i pro nedeterministické TS.

Funkce $f : \mathbb{N} \mapsto \mathbb{N}$ je *prostorově zkonstruovatelná*, pokud existuje TS tak, že pro vstup $n \in \mathbb{N}$ (reprezentovaný řetězcem 0^n) na pásce vyznačí prvních $f(n)$ políček pásky a zastaví. Přitom má pamětovou složitost $f(n)$.

Věta 1: Vztahy mezi základními třídami

Pro funkce $s(n) \geq \log(n)$ a $t(n) \geq n$ platí:

(i) $\mathbf{DTIME}(t(n)) \subseteq \mathbf{NTIME}(t(n))$ a $\mathbf{DSPACE}(s(n)) \subseteq \mathbf{NSPACE}(s(n))$

(ii) $\mathbf{DTIME}(t(n)) \subseteq \mathbf{DSPACE}(t(n))$ a $\mathbf{NTIME}(t(n)) \subseteq \mathbf{NSPACE}(t(n))$.

(iii) $\mathbf{DSPACE}(s(n)) \subseteq \mathbf{DTIME}(2^{O(s(n))})$ a $\mathbf{NSPACE}(s(n)) \subseteq \mathbf{NTIME}(2^{O(s(n))})$

(iv) $\mathbf{NTIME}(t(n)) \subseteq \mathbf{DSPACE}(t(n))$

(v) $\mathbf{NSPACE}(s(n)) \subseteq \mathbf{DTIME}(2^{O(s(n))})$

Důkaz. (i) Deterministický TS můžeme chápat jako spec. případ nedeterministického TS.

(ii) TS (a NTS v každé své výpočetní větvi) může pomocí jednoho kroku navštívit nanejvýš jedno nové políčko pásky.

(iii) Nerovnosti plynou z (i), (iv) a (v). Lze je ovšem dokázat i pomocí počítání kroků (viz poznámka za důkazem).

(iv) Ukážeme, jak NTS M s časovou složitostí $f(n)$ můžeme simulovat pomocí TS N s paměťovou složitostí z $O(f(n))$.

N provede průchod do hloubky výpočetním stromem M, přičemž pokud narazí na přijímací konfiguraci, tak také přijímá. N funguje rekurzivně, přičemž jedno rekurzivní zavolání probíhá takto: Předpokládejme, že se simulovaný stroj M nachází v konfiguraci c (tuto konfiguraci má N uloženu někde na pásce, její velikost je seshora omezena $O(f(n))$), pak

- pokud je c přijímací, N přijímá
- pro každou konfiguraci c' dostupnou z c (v TS M) pomocí jednoho kroku spustí nové rekurzivní volání.

První zavolání spustíme s počáteční konfigurací. Hloubka rekurze je omezena $f(n)$, při naivní implementaci si musíme v každé úrovni rekurze pamatovat aktuální konfiguraci c, což by vedlo k pamětové složitosti $O(f(n)^2)$. Místo toho tedy použijeme následující trik.

Nechť k je maximální počet nedeterministických voleb, které má M k dispozici. Tyto volby můžeme nějak lineárně uspořádat (např. lexikograficky) a tím pádem konkrétní volbu můžeme reprezentovat symbolem z vhodné k-prvkové abecedy. Každá výpočetní větev M si přirozeně odpovídá s řetězcem nedeterministických voleb, a tedy i s řetězcem nad zmiňovanou k-prvkovou abecedou. N si proto v rekurzivních voláních nemusí pamatovat aktuální konfiguraci. Místo toho si globálně pamatuje řetězec w nedeterministických voleb takový, že když stroj M simuluje z počáteční konfigurace a nedeterministické volby vybírá podle w, dostane se do aktuální konfigurace (kterou zrovna navštěvuje ve stromu konfigurací stroje M). Rekurzivní volání pak N provádí tak, že w rozšíří o další symbol odpovídají právě zkoumané nedet. volbě. Uvnitř tohoto rekurzivního volání si spočítá aktuální konfiguraci M tak, že jej simuluje a řídí se přitom takto rozšířeným w. Při návratu z rekurzivního zavolání pak w zase o jeden symbol zkrátí.

Pamětová složitost je nově $O(f(n))$, protože nám stačí místo pro w a jednu konfiguraci stroje M.

(v) K nedeterministickému TS M s paměťovou složitostí $f(n)$ vytvoříme determinitistický TS N s časovou složitostí $2^{O(f(n))}$ tak, že $L(M) = L(N)$. Nejdříve předpokládejme, že $f(n)$ je prostorově zkonstruovatelná. Tento předpoklad později odstraníme.

TS N na pásku zapíše všechny konfigurace stroje M, které odpovídají $f(n)$ políčkům pásky (velikost jedné konfigurace je v $O(f(n))$, konstanta uvnitř asymptotické notace závisí na počtu stavů stroje M a na velikosti páskové abeced stroje N). K tomu potřebuje umět zkonstruovat $f(n)$ a je tedy potřeba, aby tato fce byla prostorově zkonstruovatelná. Počet konfigurací, které zapíše je omezen $2^{O(f(n))}$ (jejich počet je totiž omezen $d^{f(n)}$ pro nějakou konstantu d, která závisí na M). Zapsat všechny tyto konfigurace na pásku proto trvá také nejvýše $O(2^{O(f(n))})$ kroků.

Poté N opakovaně prochází všechny konfigurace a značí si k nim, jestli jsou dosažitelné z počáteční konfigurace. Na začátku jsou všechny konfigurace, mimo počáteční, označeny jako nedosažitelné. Pak v jednom průchodu N označí všechny doposud neoznačené konfigurace, ke kterým se lze dostat jedním výpočetním krokem M z již označené konfigurace, také jako dosažitelné. V momentě, kdy N narazí na dosažitelnou přijímací

konfiguraci, pak přijíma. Pokud při průchodu konfiguracemi neoznačí jako dosažitelnou žádnou novou konfiguraci, zamítá. Jeden průchod konfiguracemi lze provést v čase polynomicky závislém na $O(2^{O(f(n))})$ a průchodů je nejvýše $O(2^{O(f(n))})$, proto je celková složitost také $O(2^{O(f(n))})$.

Zbývá odstranit předpoklad prostorové zkonstruovatelnosti $f(n)$. Tento předpoklad můžeme odstranit tak, že budeme výše popsaný algoritmus spouštět postupně pro hodnoty $f(n) = 1, 2, 3 \ldots$. Vždy, když při průchodu konfiguracemi M poznáme, že je možné dostat se jedním krokem M z dosažitelné konfigurace do konfigurace, která potřebuje více paměti, tak restartujeme celý algoritmus s hodnotou $f(n)$ o jedna vyšší.

□

TS, který si počítá počet kroků TS (nebo NTS) s pamětovou složitostí $f(n)$ lze upravit tak, aby počítal počet kroků, které již provedl a přitom měl pamětovou složitost $O(f(n))$. Protože počet konfigurací odpovídajících $f(n)$ políčků paměti je seshora omezen $d^{f(n)}$ pro vhodnou konstantu d. Pokud tedy TS provedl již více než $d^{f(n)}$ kroků, musel některou konfiguraci navštívit vícekrát a tedy cyklí. V takovém případě můžeme zamítat a nezměníme tím přijímaný jazyk (samozřejmě, pokud TS vždy zastaví, pak nikdy neprovede více $d^{f(n)}$ kroků). Počet provedených kroků počítáme v proměnné, jejíž hodnota je nejvýše $d^{f(n)}$ a pro její uložení stačí $\log d^{f(n)} = O(f(n))$ políček pásky. Takto upravený stroj také zastaví po nejvýše $d^{f(n)}$ krocích, a tedy jeho existence je alternativním důkazem bodu (iii) předchozí věty.

1.1 (Deterministické) hierarchické věty

Věta 2: O separaci DSPACE

Nechť $s(n)$ je prostorově zkonstruovatelná funkce. Potom existuje $L \in \mathbf{DSPACE}(s(n))$ tak, že $L \notin \mathbf{DSPACE}(s'(n))$ pro všechny $s'(n) \in o(s(n))$.

Důkaz. Najdeme TS M s pamětovou složitostí $f(n)$ (kde f je prostorově konstruovatelná) tak, že pro každý TS N s pamětovou složitostí v $o(f(n))$ platí $L(M) \neq L(N)$. Uděláme to diagonální metodou.

Využijeme následujících předpokladů o Turingových strojích (bez ztráty na obecnosti).

- zafixujeme abecedu $\Sigma = \{0, 1\}$.
- každý řetězec $w \in \{0, 1\}^+$ kóduje nějaký TS: o řetězcích, které neodpovídají dohodnutému zakódování TS předpokládáme, že kódují TS, který v prvním kroku zamítá.
- v kódování TS ignorujeme 0 na začátku. Každý TS má tedy nekonečně mnoho různých zakódovaní, které se liší počtem 0 na začátku řetězce.

Pro $w \in \{0, 1\}^+$ označujeme $\#w$ číslo, jehož je w binárním zápisem (všimněte si, že zde se také ignorují 0 na začátku w). Pomocí M_i značíme TS, jehož zakódováním je w a $\#w = i$.

Nyní vytvoříme požadovaný TS M. Ten pro vstup x délky $|x| = n$

1. Označí $f(n)$ políček pásky (f je prostorově konstruovatelná).

2. Simuluje M_i, pro $i = \#x$, na vstupu x
   - pokud by při simulaci potřeboval více než $f(n)$ políček pásky, tak zamítá.
   - pokud simulace zastaví (viz poznámka níže), tak pokud M_i přijímá, M zamítá. Jinak M přijímá.

Komentáře k simulaci u bodu 2:

- Nemusíme se obávat toho, že M_i pro x cyklí (a tím pádem by cyklil i M). M totiž může počítat počet kroků, které už M_i při simulaci provedl a detekovat zacyklení.
- Pro každý TS M_i běžící s pam. slož $o(f(n))$ platí, že existuje jeho dostatečně dlouhé zakódování x tak, že při simulaci stroji M nedojde paměť. Zakódovaní TS totiž můžeme na začátku vycpat nulami.

M pracuje očividně s paměťovou složitostí $f(n)$. Pro každý TS N s pamětovou složitostí v $o(f(n))$, existuje jeho dostatečně velké zakódování x (vycpané 0 na začátku) tak, že se M a N v přijímání/zamítání x liší.

Závěrečná poznámka: M potřebuje k simulaci N navíc $\log|N|$ paměti, kde $|N|$ je délka zakódování N. Pro účel simulace ovšem bereme zakódování N bez 0 na jeho začátku a můžeme se tak na jeho délku brát jako na konstatu (respektive víme, že můžeme zakódování N na začátku dostatečně vycpat 0 tak, aby měl M pro simulaci dostatek paměti).

□

Věta 3: O separaci DTIME

Nechť $t(n)$ je časově zkonstruovatelná funkce, $t(n) \geq n$. Potom existuje $L \in \mathbf{DTIME}(t(n))$ tak, že $L \notin \mathbf{DTIME}(t'(n))$ pro všechna $t'(n)$ taková, že $t'(n)\log t'(n) = o(t(n))$.

Důkaz. Analogický předchozí větě. Logaritmický faktor je cena, kterou platíme za simulaci. □

1.2 Savitchova věta

Věta 4: Savitchova

Nechť $s(n)$ je funkce taková, že $s(n) \geq n$. Potom platí

$$\mathbf{NSPACE}(s(n)) \subseteq \mathbf{DSPACE}(s(n)^2).$$

Důkaz. Vezmeme $L \in \mathbf{NSPACE}(s(n))$ a dokážeme, že $L \in \mathbf{DSPACE}(s(n)^2)$. Protože $L \in \mathbf{NSPACE}(s(n))$ existuje NTS M s paměťovou složitostí $f(n) \in O(s(n))$. Sestavíme TS, který bude rozhodovat L s pamětovou složitostí v $O(f(n)^2)$, a tím pádem i v $O(s(n)^2)$.

Navrhneme deterministický algoritmus CAN-YIELD, který má vstup α, β a t, kde α, β jsou konfigurace stroje M obsahující $f(n)$ políček pásky (ve skutečnosti jsou tedy konfigurace dlouhé $f'(n) = f(n) + c$, kde c je konstance závisející na M, zejména na počtu jeho stavů) a t je počet kroků. Algoritmus rozhodne, zdali se M může dostat z konfigurace α do konfigurace β s pomocí maximálně t kroků (přičemž předpokládáme, že M nebude potřebovat více než $f(n)$ paměti).

Algoritmus CAN-YIELD pro vstup (α, β, t):

1. pokud $t = 0$ zkontrolujeme $\alpha = \beta$. Pokud ano, přijímáme, jinak zamítáme
2. pokud $t = 1$ zkontrolujeme, jestli z α umí M přejít do β pomocí jednoho kroku (to lze zjistit z přechodové funkce M, kterou máme zadrátovanou napevno v algoritmu).
3. Jinak iterujeme přes všechny konfigurace γ (obsahující $f(n)$ políček pásky) a pro každou z nich rekurzivně zavoláme CAN-YIELD$(\alpha, \gamma, \lfloor t/2 \rfloor)$ a CAN-YIELD$(\gamma, \beta, \lceil t/2 \rceil)$. Pokud obě volání přijímají, tak přijímáme (a cyklus končí).
4. Zamítáme.

Nyní si uvědomíme, že bez ztráty na obecnosti můžeme předpokládat, že M má unikátní přijímací konfiguraci ACCEPT (např. M před přijetím smaže pásku, přejede s čtecí hlavou nad její začátek a a přejde do unikátního přijímacího stavu). Protože délka konfigurací M (pro vstup x délky $|x| = n$) je omezena $f'(n)$, maximální délka výpočtu M je $d^{f'(n)}$ pro vhodnou konstantu d (která závisí pouze na M). Pokud tedy pro vstup x (příslušnou počáteční konfiguraci označíme START) zavoláme CAN-YIELD(START, END, $d^{f'(n)}$), a přijímáme právě když toto zavolání přijímá, máme TS, který rozhoduje L.

Pro jedno zavolání CAN-YIELD potřebujeme $O(f(n))$ paměti (ukládáme pouze konstantní počet konfigurací, jejichž velikost je omezena $O(f(n))$). Díky tomu, že při rekurzivním volání půlíme třetí argument, je maximální hloubka rekurze rovna $\log_2 d^{f'(n)} \in O(f'(n)) \subseteq O(f(n))$, celkově tedy potřebujeme $O(f(n)^2)$ paměti.

Poznámka na závěr: Tiše jsme předpokládali, že $f(n)$ je prostorově zkonstruovatelná. Tento předpoklad můžeme odstranit tak, že budeme výše popsaný algoritmus spouštět postupně pro hodnoty $f(n) = 1, 2, 3 \ldots$. Vždy, když v CAN-YIELD poznáme, že bychom potřebovali více paměti pro konfigurace (například máme dosažitelnou konfiguraci M s hlavou na posledním políčku vpravo a podle přechodové funkce M se s ní lze posunout ještě více doprava), tak restartujeme celý algoritmus hodnotou $f(n)$ o jedna vyšší. □

1.3 Immermanova-Szelepcsenyiho věta

Pro třídu problémů K definujeme třídu $\mathsf{co}K$ pomocí: $L \in \mathsf{co}K$ právě když $\bar{L} \in K$.

Věta 5: Immermanova-Szelepcsenyiho věta

Nechť $s(n) \geq \log n$. Potom **NSPACE**$(s(n))$ = **coNSPACE**$(s(n))$.

Důkaz. Vezmeme $L \in$ **NSPACE**$(s(n))$ a dokážeme, že $\bar{L} \in$ **coNSPACE**$(s(n))$. Protože $L \in$ **NSPACE**$(s(n))$ existuje NTS M s paměťovou složitostí $f(n) \in O(s(n))$. Sestavíme NTS N, který bude rozhodovat $\bar{L}$ s pamětovou složitostí v $O(f(n))$, a tím pádem i v $O(s(n))$.

Využijeme následujících pozorování.

- Můžeme předpokládat, že M před přijetím smaže obsah pásky, posune hlavu na první políčko pásky a přejde do unikátního přijímacího stavu. Získáme tak unikátní přijímcí konfiguraci, kterou označíme ACCEPT.

- Konfigurace M jsou dlouhé maximálně $f'(n) \in O(f(n))$ ($f(n)$ políček pásky plus navíc je potřeba zaznamenat aktuální stav, ale k tomu stačí řetezec konstantní délky). M se může dostat do maximálně $d^{f'(n)}$ konfigurací, pro vhodnou konstantu d, která závisí na M (počtu stavů, velikosti abeced atd.). Z toho plyne, že při přijímání provede M maximálně $d^{f'(n)}$ kroků.

Předpokládejme vstup x o velikosti $|x| = n$, označme počáteční konfiguraci M (pro x) START. Uvažme množinu

$$A_m = \{\alpha \mid \alpha \text{ je konfigurace dosažitelná ze START pomocí maximálně } m \text{ kroků}\}.$$

Vidíme, že $A_0 = \{\text{START}\}$ a tedy $|A_0| = 1$. Následující nedeterministický algoritmus spočítá $|A_{m+1}|$, pokud známe $|A_m|$.

1. Proměné `result` a `counter` inicializujeme na 0.
2. Pro každou konfiguraci α stroje M (odpovídající $f(n)$ políčkům pásky) provedeme následující:
   (a) proměnou `reachable` inicializujeme na 0.
   (b) Pro každou konfiguraci β stroje M (odpovídající $f(n)$ políčkům pásky) provedeme následující:
   - nedeterministicky vybereme a simulujeme maximálně m kroků dlouhý výpočet stroje M z konfigurace START, označme konfiguraci, ve které tento výpočet končí γ.
   - pokud $\beta = \gamma$, pak inkrementujeme proměnnou `counter` o 1.
   - pokud se lze jedním výpočetním krokem stroje M dostat z konfigurace β do konfigurace α, nastavíme proměnnou `reachable` na 1.

   (c) Je-li hodnota proměnné `counter` různá od $|A_m|$, zamítáme. (*Toto je koncepčně důležitý test, viz vysvětlení níže.*)
   (d) Je-li hodnota proměnné `reachable` rovna 1, inkrementujeme proměnnou `result`
3. Výsledek je v proměnné result.

Idea algoritmu je následující: pro kažkou konfiguraci α testujeme, jestli je dosažitelná pomocí max $m + 1$ kroků z konfigurace START. Děláme to tak, že iterujeme přes všechny konfigurace β dosažitelné pomocí max m kroků. Zde ale jejich dosažitelnost testujeme tak, že se nedeterministky vybereme výpočet s max m kroky a provedeme ho. Existuje tak nebezpečí, že tento výpočet nevybereme správně (nevede k β, i když je β pomocí max m kroků dosažitelná). K předejití tohoto slouží proměnná counter, ve které si počítáme, kolik výpočtů jsme nedeterministicky vybrali správně. Pokud se tato hodnota nerovná $|A_m|$, pak víme, že jsme někde vybrali špatně a zamítáme. Algoritmus potřebuje $O(f(n))$ paměti, protože uchovává dvě konfigurace stroje M, simuluje výpočet M, a uchovává dvě čísla omezená seshora $d^{f'(n)}$, která lze (ve vhodné abecedě) reprezentovat pomocí $O(f'(n))$ znaků.

Předpokládejme nyní, že máme k dispozici $|A_{d^{f'(n)}}|$, které jsme s pamětovou složitostí $O(f(n))$ spočítali pomocí opakované aplikace předchozího algoritmu. Následující algoritmus pak otestuje, jestli je ACCEPT dosažitelná pomocí max $d^{f'(n)}$ kroků ze START, a pokud není, tak přijímá (jinak zamítá). Algoritmus tak rozhoduje $\bar{L}$. Algoritmus funguje následovně:

1. Proměnnou `counter` inicializujeme na 0.
2. Pro každou konfiguraci α stroje M (odpovídající $f(n)$ políčkům pásky) provedeme následující:
   (a) nedeterministicky vybereme a simulujeme maximálně $d^{f'(n)}$ kroků dlouhý výpočet stroje M z konfigurace START, označme konfiguraci, ve které tento výpočet končí γ.
   (b) pokud se γ a α rovnají, inkrementujeme `counter` o 1. Pokud se navíc α rovná ACCEPT, zamítáme.
3. Pokud se proměnná `counter` nerovná $|A_{d^{f'(n)}}|$, zamítáme.
4. Přijímáme.

Vidíme, že v algoritmu se opakuje trik z předchozího algoritmu zajišťující správnost nedeterministických voleb. V kroku 2 opravdu musíme projít všechny konfigurace (a nestačí jenom přijímací), protože pokud v kroku 2 nezamítneme, musíme ještě v kroku 3 zkontrolovat, že se to nestalo jen kvůli tomu, že jsme nedeterministicky vybrali výpočty M nevedoucí k ACCEPT, ale ACCEPT je přesto dosažitelný.

Navíc vidíme, že přijímáme, pouze pokud není v max $d^{f'(n)}$ krocích dosažitelná konfigurace ACCEPT. Také je vidět, že potřebujeme pouze $O(f'(n))$ paměti, to je podobné jako u předchozího algoritmu. Celkově tedy N, který konstruujeme,

1. spočítá a uloží si $|A_{d^{f'(n)}}|$,
2. spustí předchozí algoritmus.

Poznámka na závěr: tiše jsme předpokládali, že $f(n)$ je prostově zkonstruovatelná. Tento přepoklad jde obejít standardním argumentem, který jsme viděli v předchozích důkazech. □

Literatura.

Kapitola je založena převážně na [2]: kapitoly 1 až 4. Důkaz Savitchovi věty byl čerpán z [1], kapitola 8.1.

2 Třídy P a NP

Nejdříve definujeme třídy **P** a **NP**:

$$\mathbf{P} = \bigcup_{k>0} \mathbf{DTIME}(n^k)$$
$$\mathbf{NP} = \bigcup_{k>0} \mathbf{NTIME}(n^k)$$

Vidíme, že triviálně platí $\mathbf{P} \subseteq \mathbf{NP}$ (plyne to z Věty 1 (i)).

Jazyk L je polynomicky verifikovatelný, pokud existuje polynom q a DTS V bežící v polynomickém čase tak, že $x \in L$ právě když existuje řetězec c délky nejvýše $q(|x|)$ tak, že $V(x, c) = 1$ (tj. V přijímá pro vstup x, c). DTS V říkáme *verifikátor* a c říkáme *certifikát* nebo *důkaz* toho, že x patří do L.

Věta 6: Nedeterminismus vs. verifikovatelnost

$L \in \mathbf{NP}$ právě když L je polynomicky verifikovatelný.

Důkaz. Implikace zleva doprava. Předpokládejme $L \in \mathbf{NP}$. Existuje tak NTS M tak, že $L = L(M)$. V přechodové funkci NTS M najdeme maximální počet nedeterministických voleb, které má M při provádění jednoho kroku k dispozici, označme si toto číslo k. Posloupnost voleb, které M v některé větvi výpočtu provede můžeme tedy reprezentovat pomocí řetězce nad k-prvkovou abecedou (v každém pravidle M si volby očíslujeme tak, aby každá odpovídala některém znaku nad touto k-prvkovou abecedou).

Sestavíme verifikátor V. Ten pro vstup w, c:

1. Simuluje M, přičemž c interpretuje jako posloupnost voleb stroje M, jak jsme diskutovali výše. Simuluje tak výpočet M na výpočetní větvi určené c.
2. Pokud M v této simulaci přijímá, pak V také přijímá. Jinak V zamítá.

Vidíme, že pokud $x \in L$ pak existuje přijímací výpočetní větev NTS M, tj. existuje posloupnost nedeterministických voleb c stroje M, které tuto výpočetní větev determinují, a tedy V pro vstup x přijímá. Pokud $x \notin L$, pak žádná posloupnost nedeterministických voleb neodpovídá přijímací větvi výpočtu M a verifikátor V pak zamítá pro každou dvojici (x, c).

Protože M pracuje v polynomickém čase, má každá jeho výpočetní větev polynomickou délku a tedy i V pracuje v polynomickém čase.

Implikace zprava doleva. Předpokládáme, že L je polynomicky verifikovatelný, tedy existuje verifikátor V pracující v polynomickém čase. Sestavíme NTS M tak, že $L = L(M)$. Tento stroj pro vstup x

1. Nedeterministicky vytvoří řetězec c (délka tohoto řetězce je omezena: pokud V pracuje v čase n^k, stačí $|c| = n^k$.)
2. Simuluje V pro vstup (x, c). Pokud V přijímá, M také přijímá. Jinak M zamítá.

Pokud existuje c' tak, že V přijímá (x, c'), pak M toto c' v kroku jedna zvolí a simulace V skončí přijetím. Pokud takové c neexistuje, pak simulace V pro každou volbu v kroku jedna skončí zamítnutím. Platí tedy, že $L = L(M)$.

□

Příklady problémů z NP

- $SAT = \{\varphi \mid \varphi$ je splnitelná formule výrokové logiky v konjunktivní normální formě$\}$. Jako certifikát toho, že fle φ je splnitelná lze použít ohodnocení e výrokových proměnných, pro které je φ pravdivá. Verifikátor tedy pro dvojici φ, e spočítá pravdivost φ při ohodnocení e. To lze provést v polynomickém čase vzhledem k velikosti φ. Pokud φ není splnitelná, pak žádné ohodnocení proměnných, pro který by byla pravdivá, neexistuje.
- 3SAT je podmnožina SAT obsahující pouze formule s klausulemi obsahujícími právě 3 literály.
- $CLIQUE = \{(G, k) \mid G$ je neorientovaný graf obsahující kliku s alespoň k uzly$\}$. Klika v grafu G je jeho úplný podgraf. Jako certifikát můžeme použít množinu vrcholů, která tvoří v G kliku potřebné velikosti. Ověřit, že skutečně jedná o kliku, lze v polynomickém čase.

2.1 NP-obtížnost a NP-úplnost

Funce $f : \Sigma^* \to \Sigma^*$ je *spočitatelná v polynomickém čase*, pokud existuje TS, který pro vstup w zastaví po polynomickém počtu kroků (vzhledem k $|w|$) a na pásce zůstane řetězec $f(w)$.

Jazyk A je v *polynomickém čase redukovatelný* na B, značíme $A \leq_p B$, pokud existuje funkce f spočitatelná v polynomickém čase tak, že $x \in A$ právě když $f(x) \in B$.

Věta 7

$A \leq_p B$ a $B \in P$ implikuje $A \in P$.

Důkaz. Pro A existuje následující TS s pol. čas. slož

1. Pro vstup x spočítá v polynomickém čase $f(x)$, kde f je funkce existující díky $A \leq_p B$. Existuje polynom p_1 omezující časovou složitost výpočtu $f(x)$ z x a pro který tak máme $|f(x)| \leq p_1(|x|)$.
2. Pro B existuje TS pracující v polynomickém čase, předp. že ten je seshora omezen polynomem p_2. Tento TS spustíme se vstupem $f(x)$ a vrátíme jeho výsledek. Výpočet trvá maximálně $p_2(p_1(|x|))$ kroků a je tedy polynomicky dlouhý vzhledem k $|x|$.

□

Důsledkem předchozí věty je: $A \notin P$ implikuje $B \notin P$.

Věta 8: Tranzitivita polynomické redukce

$A \leq_p B$ a $B \leq_p C$ implikuje $A \leq_p C$.

Důkaz. Korektnost plyne z tranzitivity logické spojky *právě když*. Složitost: Složením dvou funkcí spočitatelných v polynomickém čase dostaneme funkci spočitatelnou v polynomickém čase (viz důkaz předchozí věty). □

Jazyk L je **NP**-těžký, pokud pro každý $L' \in NP$ máme $L' \leq_p L$. L je **NP**-úplný, pokud je **NP**-těžký a $L \in$ **NP**. Rychle vidíme, že pokud je **NP**-těžký problém v **P**, pak **P** = **NP**. Z tohoto pohledu jsou **NP**-úplné problémy v rámci **NP** nejtěžší.

Věta 9

Pokud je A **NP**-těžký a $A \leq_p B$, pak i B je **NP**-těžký.

Důkaz. Plyne z tranzitivity $\leq_p$. □

2.2 Příklady redukcí

Věta 10

$SAT \leq_p 3SAT$

Důkaz. Nechť $\varphi = F_1 \wedge F_2 \wedge \cdots \wedge F_k$ je formule v CNF (tj. instance SAT problému). Ukážeme, jak libovolnou klausuli F_i převedeme na formuli $\varphi_i = C_1^i \wedge C_2^i \ldots$ takovou, že: (a) C_j^i jsou klausule s právě třemi literály; (b) φ_i je pravdivá, právě když F_i je pravdivá. Pak zjevně φ je pravdivá, právě když $\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_k$ je pravdivá.

Označme klausuli, kterou budeme transformovat, pomocí F_i. Pak jsou tři možnosti:

1. F_i má právě tři literály. V tomto případě vezmeme $\varphi_i = F_i$
2. F_i má méně než tři literály. Pak stačí jeden z literálů obsažených v F_i zdvojit (nebo ztrojit), získáme tak klausuli se třemi literály a použijeme předchozí bod.
3. F_i má více než 3 literály. Přepokládejme tedy, že

$$F_i = (l_1 \vee l_2 \vee \cdots \vee l_n) \tag{1}$$

Pak

$$\varphi_i = (l_1 \vee l_2 \vee z_1) \wedge (\neg z_1 \vee l_2 \vee z_2) \wedge (\neg z_2 \vee l_3 \vee z_3) \wedge \cdots \wedge (\neg z_{n-2} \vee l_{n-1} \vee l_n), \tag{2}$$

kde $z_1, z_2 \ldots z_{n-2}$ jsou nové (doposud nepoužité) výrokové proměnné.

V případech 1. a 2. zjevně platí, že F_i je pravdivá, právě když φ_i je pravdivá. Rozebereme případ 3. Pokud je F_i pravdivá, pak exituje literál l_j, který splní jednu klausuli ve formuli (2), zbylé klausule pak jde splnit pomocí vhodného ohodnocení proměnných $z_1, \ldots, z_{n-2}$ (čtenář snadno ověří jako cvičení). Pokud je naopak pravdivá (2), pak alespoň jeden literál l_j musí být pravdivý, protože (2) nelze splnit pouze pomocí vhodného ohodnocení proměnných $z_1, z_2, \ldots z_{n-2}$. To plyne z toho, že sousední klausule v (2) obsahují vždy proměnnou (klausule vlevo) a její negaci (klausule vpravo).

Právě popsaný převod klausule na konjunkci klausulí lze jistě provést v polynomickém čase (závislém na počtu literálů v klausuli) a tedy transformaci φ na $\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_k$ lze provést v polynomickém čase. □

Věta 11

$3SAT \leq_p CLIQUE$

Důkaz. Nechť $\varphi = F_1 \wedge F_2 \cdots \wedge F_m$. Sestavíme graf G a číslo k.

Ke každé klausuli $F_i = (l_1^i \vee l_2^i \vee l_3^i)$ sestavíme graf G_i s třemi vrcholy, první označíme pomocí l_1^i, druhý pomocí l_2^i a třetí pomocí l_3^i. Vrcholy grafu G jsou právě všechny vrcholy v G_i pro $i = 1, \ldots, m$. Zbývá dodat hrany. Přidáme hrany spojující vrcholy (u, v) pro které platí:

1. $u \in G_i$, $v \in G_j$ a $i \neq j$ (nepatří do stejného grafu G_i vytvořeného nahoře).
2. je-li u označen literálem l a v literálem l' pak dvojice l, l' netvoří výrokovou proměnnou a její negací.

Nastavíme $k = m$. Rychle vidíme, že graf G jde sestavit v polynomickém čase.

Pokud existuje ohodnocení proměnných, pro které je φ pravdivá, je v každé klausuli φ alespoň jeden literál pravdivý. Vytvoříme množinu C uzlů v G: pro $i = 1, \ldots, m$ přidáme do C jednu vrchol, který je označem pravdivým literálem. Nyní si všimneme, že pro libovolné dva vrcholy v C platí, že literály, kterými jsou označeny, nejsou výrokovou proměnnou a její negací (protože takové literály nemohou být současně pravdivé). Proto mezi všemi vrcholy v C jsou v grafu G hrany, a C tak tvoří kliku s k uzly.

Předpokládejme, že v G existuje klika C s alespoň k uzly. Protože uzly v rámci žádného grafu G_i spolu nejsou spojeny hranami, musí mít tato klika právě k uzlů, z každého z grafů $G_1, \ldots, G_k$ jeden. Pokud ohodnotíme výrokové proměnné formule φ tak, že všechny literály, kterými jsou označeny vrcholy z C, jsou pravdivé, pak se v každé klausuli vyskytuje jeden pravdivý literál a φ je tak pravdivá. Takové ohodnocení přitom najít můžeme, protože vrcholy označené proměnnou a její negací nejsou spolu spojeny hranou a nemohou být současně v C. □

2.3 Cook-Levinova věta a její důkaz

Větu dokážeme s pomocí booleovských obvodů. Hodnotu 0 idenfikujeme s pravdivostní hodnotou nepravda, 1 s hodnotou pravda.

Připomeňme pojem n-ární booleovské funkce $f : \{0,1\}^n \mapsto \{0,1\}$. Každou takovou funkci můžeme reprezentovat tabulkou s 2^n řádky, formulí výrokové logiky s n proměnnými (kterou můžeme rutinním postupem vytvořit z tabulky). Booleovskou funkci můžeme také reprezentovat booleovským obvodem.

Booleovský obvod je orientovaný acyklický graf $G = (V, E)$. Vrcholům říkáme brány. Každá brána má label z množiny $\{\vee, \wedge, \neg, 0, 1\} \cup \{x_1, x_2, x_3 \ldots\}$, kde $x_1, x_2, \ldots, x_n$ jsou vstupní proměnné. Brány s labely $0, 1, x_1, \ldots, x_n$

mají *indegree* (indegree je počet hran, které do bráno vedou z jiných bran) roven 0, brány s labelem $\neg$ mají indegree 1, a brány s labely $\vee, \wedge$ mají indegree 2. Jedna brána s outdegree 0 (outdegree je počet hran, které vedou z brány do jiných bran) je označena jako výstupní.

Pro ohodnocení proměnných $e : \{x_1, \ldots, x_n\} \mapsto \{0, 1\}$ je *pravdivost* $T(q)$ *brány* q definována:

- pokud má q indegree 0, pak

$$T(q) = \begin{cases} 0 & q \text{ má label } 0 \\ 1 & q \text{ má label } 1 \\ e(x_i) & q \text{ má label } x_i \end{cases}$$

- pokud má q indegree 1, a do q vede hrana z brány h, pak $T(q) = \neg T(h)$.
- pokud má q indegree 2, a do q vedou hrany z bran h_1, h_2, pak

$$T(q) = \begin{cases} T(h_1) \vee T(h_2) & q \text{ má label } \vee \\ T(h_1) \wedge T(h_2) & q \text{ má label } \wedge \end{cases}$$

Pravdivost booleovského obvodu je rovna pravdivosti jeho výstupní brány.

Funkci $g : \{0, 1\}^n \mapsto \{0, 1\}^m$ můžeme považovat za m n-árních booleovských funkcí a reprezentovat ji obvodem (který je např. sjednocením m okruhů reprezentujících jednotlivé n-ární funkce, přičemž tyto sdílejí n vstupních bran; lze si také představit, že sdílejí více bran). Takový obvod má m výstupních bran.

Funkci $g : \Sigma^k \mapsto \Sigma$ pro obecnou abecedu Σ a přirozené k můžeme reprezentovat pomocí booleovské funkce $f : \{0, 1\}^{k \cdot m} \mapsto \{0, 1\}^m$, kde $m = \lceil \log_2 |\Sigma| \rceil$, a tedy i pomocí boolevoského obvodu. Stačí vzít binární reprezentaci znaků Σ pomocí m-bitových řetězců.

Uvažujeme dva problémy:

- CIRCUIT-VALUE. Vstupem je booleovský obvod bez proměnných (tj. brány s indegree 0 jsou pouze brány s labely 0,1). Chceme spočítat pravdivost tohoto obvodu.
- CIRCUIT-SAT. Vstupem je booleovský obvod s n proměnnými. Ptáme se, jestli existuje ohodnocení proměnných takové, že obvod je pro ně pravdivý.

Vidíme, že CIRCUIT-VALUE $\in$ **P** (stačí najít topologické uspořádání bran a pomocí něj spočítat jejich pravdivosti). V důsledku toho máme CIRCUIT-SAT $\in$ **NP**. Instanci CIRCUIT-SAT lze totiž s pomocí ohodnocení proměnných převést na instanci CIRCUIT-VALUE: brány odpovídající proměnným nahradíme branami s labely $0, 1$ podle daného ohodnocení proměnných. Ohodnocení proměnných tedy slouží jako certifikát.

Věta 12

CIRCUIT-SAT $\leq_p$ SAT.

Důkaz. Pro booleovský obvod C s proměnnými $x_1, \ldots, x_n$ a branami $g_1, \ldots, g_m$ vytvoříme formuli φ s proměnnými $x_1, \ldots, x_n, g_1, \ldots, g_m$ (tedy proměnnými φ jsou proměnné a brány C). Pro každou bránu obvodu C vytvoříme formuli v CNF a takto získané formule spojíme pomocí konjunkce a dostaneme tak φ. Uvažujme tedy bránu g:

- pokud má g label 1, vytvoříme formuli g, pokud má label 0, vytvoříme formuli $\neg g$.
- pokud má g label x_i, vytvoříme formuli $(\neg g \vee x_i) \wedge (g \vee \neg x_i)$. To je CNF tvar formule $(g \Leftrightarrow x_i)$.
- pokud má g label $\neg$ a vede do ní hrana z brány h, pak vytvoříme formuli $(\neg g \vee \neg h) \wedge (g \vee h)$. To je CNF tvar formule $(g \Leftrightarrow \neg h)$.
- pokud má g label $\vee$ a vedou do ní hrany z bran h_1, h_2, pak vytvoříme formuli $(\neg h_1 \vee g) \wedge (\neg h_2 \vee g) \wedge (h_1 \vee h_2 \vee \neg g)$. To je CNF tvar formule $(h_1 \vee h_2) \Leftrightarrow g$.
- pokud má g label $\wedge$ a vedou do ní hrany z bran h_1, h_2, pak vytvoříme formuli $(\neg g \vee h_1) \wedge (\neg g \vee h_2) \wedge (\neg h_1 \vee \neg h_2 \vee g)$ To je CNF tvar formule $(h_1 \wedge h_2) \Leftrightarrow g$.
- pokud je g výstupní brána, vytvoříme navíc formuli g.

Indukcí (přes topologické uspořádání bran) snadno dokážeme, že C je pravdivý pro ohodnocení proměnných e, právě když existuje doplnění tohoto ohodnocení (musíme doplnit ohodnocení proměnných $g_1, \ldots, g_m$) tak, že výsledné ohodnocení splní φ.

φ lze zkonstruovat v polynomickém čase vzhledem k velikosti (tj. počtu bran) obvodu C. □

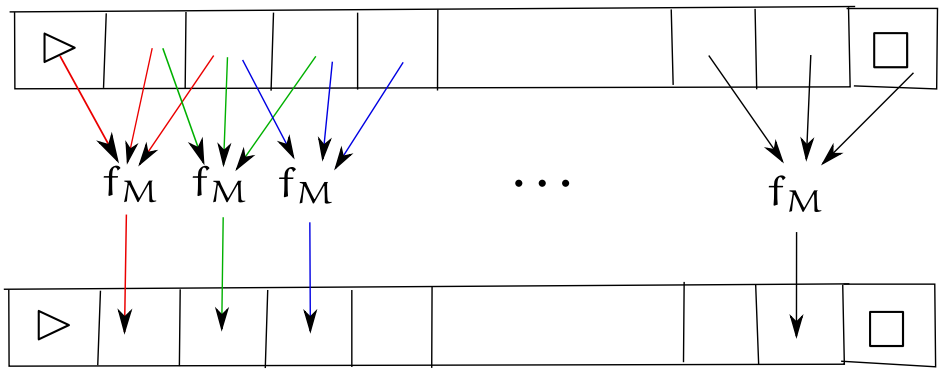
Tableau metoda. Uvažme TS M rozhodující jazyk L, přičemž pro vstup x provede vždy ostře méně než $|x|^k - 1$ kroků (pro nějaké přirozené k.) Takový TS existuje pro každý jazyk z třídy **P**. Výpočet M pro vstup x si lze představit jako posloupnost konfigurací. Tuto posloupnost zachytíme pomocí tabulky TAB s $|x|^k$ sloupci a $|x|^k - 1$ řádky (obojí číslováno od 0). Sloupec 0 tabulky vždy obsahuje speciální symbol $\triangleright$ (předpokládejem, že se nenachází páskové abecedě M). Ve zbytku řádku i v tabulce bude vždy zapsána konfigurace M po provedení prvních i kroků výpočtu (tedy na řádku 0 je počáteční konfigurace) a to následujícím způsobem:

- pokud se po provedení i kroků nachází čtecí/zapisovací hlava nad j-tým políčkem pásky a toto políčko obsahuje symbol σ, je TAB$[i, j]$ rovno (q, σ), kde q je stav, ve kterém se M nachází po provedení i kroků. Výjimkou jsou situace, kdy q je přijímací stav, to je TAB$[i, j]$ = ACCEPT, nebo kdy je q zamítací stav, to je TAB$[i, j]$ = REJECT. Předpokládáme přitom, že symboly REJECT a ACCEPT se nenachází v páskové abecedě M.
- pokud se po provedení i kroků čtecí hlava nachází nad jiným než j-tým políčkem pásky, a na j-tém políčku pásky je symbol σ, pak TAB$[i, j] = \sigma$.
- pokud M zastavil dříve než po $|x|^k - 2$ krocích, je řádek odpovídající poslední konfiguraci M zkopírován do zbývajících řádků tabulky (a tabulka je tak vyplněna celá).

Můžeme bez ztráty na obecnosti předpokládat, že M před zastavením smaže celou pásku a čtecí hlavu přesune na její začátek. Symboly REJECT a ACCEPT se tak mohou nacházet pouze ve sloupci 1. Dále si můžeme všimnout, že sloupec $|x|^k - 1$ obsahuje vždy symbol $\square$ označující prázdné políčko, to je proto, že stroj provede nejvýše $|x|^k - 2$ kroků a na $(|x|^k - 1)$-té políčko tedy nestihne nic zapsat. Platí, že pokud $x \in L$, pak $\text{TAB}[|x|^k - 2, 1] = \text{ACCEPT}$ a pokud $x \notin L$, pak $\text{TAB}[|x|^k - 2, 1] = \text{REJECT}$. Tabulka v následujícím obrázku zachycuje přijímací výpočet pro vstup $x = x_1 x_2 \dots x_n$.

$\triangleright$	(x_1, q_0)	x_2	$\dots$	x_n	$\square$	$\dots$	$\square$
$\triangleright$							$\square$
$\vdots$			$\vdots$				$\vdots$
$\triangleright$	ACCEPT						$\square$

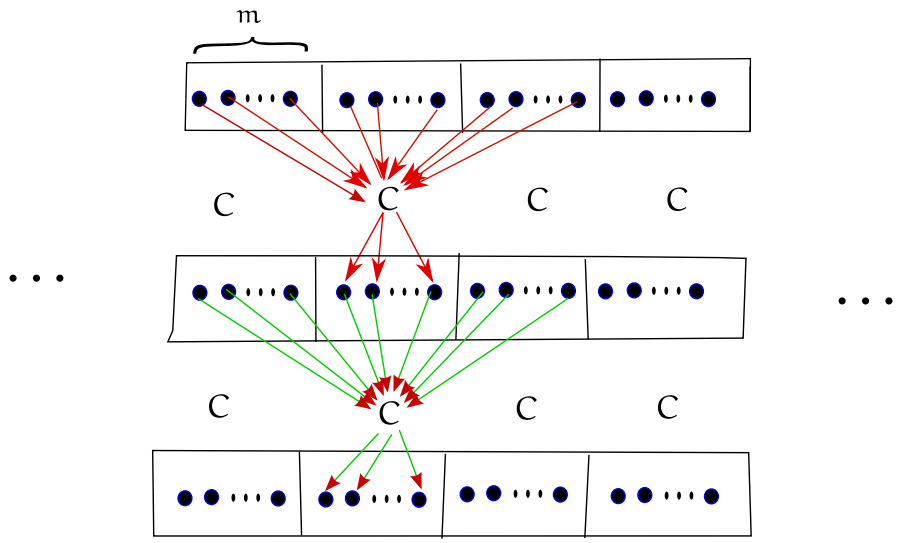
Nyní si všimneme klíčové věci: pro $0 < i, 0 < j < |x|^k - 1$ platí, že $\text{TAB}[i, j]$ závisí pouze na $\text{TAB}[i-1, j-1]$, $\text{TAB}[i-1, j]$ a $\text{TAB}[i-1, j+1]$, tedy na třech políčcích z předchozího řádku. Pokud obsah těchto políček neoznačuje pozici čtecí hlavy, pak $\text{TAB}[i, j] = \text{TAB}[i-1, j]$. V opačném případě musíme $\text{TAB}[i, j]$ vyčíst z přechodové funkce M (pozn.: to že jsme na začátku pásku poznáme z toho, že $\text{TAB}[i-1, j-1] = \triangleright$). Protože je M deterministický, je hodnota $\text{TAB}[i, j]$ jednoznačně určena. Závislost $\text{TAB}[i, j]$ na $\text{TAB}[i-1, j-1]$, $\text{TAB}[i-1, j]$ a $\text{TAB}[i-1, j+1]$ tedy můžeme vyjádřit jako funkci $f_M : \Gamma^3 \mapsto \Gamma$, kde Γ je množina všech symbolů, které se mohou vyskytnout v tabulce TAB. Celou tabulku tak můžeme spočítat opakovanou aplikací funkce f_M, přičemž první řádek je dán x, a první a poslední sloupec jsou konstantní. Viz následující obrázek.



Nyní položíme $m = \lceil \log_2 |\Gamma| \rceil$ a funkci f_M realizujeme pomocí booleovského obvodu C reprezentujícího booleovskou funkci $\{0,1\}^{3 \cdot m} \mapsto \{0,1\}^m$ (viz diskuze pod definicí booleovského obvodu). Důležité je, že počet bran C nezávisí na $|x|$, pouze na TS M (na velikosti jeho páskové abecedy a počtu jeho stavů).

Jednotlivé aplikace f_M nahradíme výpočtem pomocí obvodu C, a zapojíme tak $(|x|^k - 2) \cdot (|x|^k - 2)$ kopií C do velkého obvodu D. Obvodům počítajícím druhý řádek nastavíme vstupní brány na konstantní brány

odpovídající obsahu příslušných buněk tabulky v prvním řádku, obvod D tak nemá žádné proměnné. Symboly ACCEPT a REJECT reprezentujeme tak, že se liší v prvním bitu a za výstupní bránu obvodu D označíme tu odpovídající prvnímu bitu ve sloupci 1. Očividně platí, že $x \in L$ právě když obvod D je pravdivý. Princip zapojení obvodů je zachycen na následujícím obrázku.



Konstrukce, kterou jsme právě popsali vlastně realizuje redukci výpočtu TS (s pol. čas slož pro konkrétní vstup x) na výpočet (polynomicky velkého) booleovského obvodu (tedy na problém CIRCUIT-VALUE).

Věta 13: Cook-Levinova

SAT je **NP**-težký.

Důkaz. Protože jsme ukázali CIRCUIT-SAT $\leq_p$ SAT, stačí dokázat **NP**-težkost CIRCUIT-SAT.

Uvažujme jazyk $L \in \mathbf{NP}$ a NTS M, který jej rozhoduje v čase omezeném n^k. Bez ztráty na obecnosti můžeme předpokládat, že M má při každém kroku každého výpočtu právě dvě nedeterministiké volby. Konkrétní výpočetní větev (pro vstup x) tedy můžeme identifikovat jako bitový řetězec $c = c_1 c_2 \dots c_{n^k}$ vybraných nedetrministických voleb.

Pro fixní c je výpočet M deterministický, a šla by tedy provést konstrukce s pomocí tableau metody. Abychom ji mohli provést v naší situaci, stačí si uvědomit, že funkce f_M potřebuje jeden argument navíc, a to je právě příslušná nedeterministická volba. Pokud totiž chceme spočítat TAB$[i,j]$ na základě TAB$[i-1,j-1]$, TAB$[i-1,j]$, TAB$[i-1,j+1]$, pak (v případě, že nad některým z políček je čtecí hlava) potřebujeme vědět které ze dvou možných pravidel z přechodové funkce M máme použít.

Provedeme tedy celou konstrukci s tím, že výsledný obvod D má navíc n^k bran olabelovaných pomocí proměnných $x_1, \dots, x_{n^k}$. Ohodnocení těchto proměnných pak odpovídá výběru výpočetní větve ve výpočtu M. Očividně pak platí že existuje ohodnocení proměnných, pro které je D pravdivá, právě když existuje přijímací výpočetní větev M pro x, a tedy právě když $x \in L$. □

2.4 Třída coNP

Připomeneme, že $\mathbf{coNP} = \{\bar{L} \mid L \in \mathbf{NP}\}$. Můžeme říci, že **coNP** obsahuje problémy, které mají krátký certifikát pro odpověď NE (protože **NP** obsahuje problémy, které mají krátký certifikát pro odpověď ANO).

Z definice redukce v polynomickém čase vidíme, že $L_1 \leq_p L_2$ právě když $\overline{L_1} \leq_p \overline{L_2}$ a tedy **coNP**-težké problémy jsou právě doplňky **NP**-težkých problémů.

Není známo, zda $\mathbf{NP} = \mathbf{coNP}$. Vztah k **P** vs **NP** problému je následující:

- $\mathbf{P} = \mathbf{NP}$ implikuje $\mathbf{NP} = \mathbf{coNP}$, protože **P** je uzavřená na doplněk (stejně jako všechny deterministické třídy).
- důsledkem předchozího bodu je, že pokud $\mathbf{NP} \neq \mathbf{coNP}$, pak $\mathbf{P} \neq \mathbf{NP}$.

Věta 14

Pokud existuje $L \in \mathbf{NP}$, který je **coNP**-těžký, pak $\mathbf{NP} = \mathbf{coNP}$.

Důkaz. Vybereme $L' \in \mathbf{coNP}$. L je **coNP**-těžký a proto $L' \leq_p L$ (potřebnou funkci označme f). Existuje tedy NTS rozhodující L' v polynomickém čase: pro vstup x spočítá $f(x)$ a pro výsledek spustí NTS pracující v pol. čase, který rozhoduje L (připomeňme, že $L \in \mathbf{NP}$.) Máme tak $L' \in \mathbf{NP}$ a tedy $\mathbf{coNP} \subseteq \mathbf{NP}$. Odtud plyne $\mathbf{NP} \subseteq \mathbf{coNP}$ a tedy $\mathbf{NP} = \mathbf{coNP}$. □

Snadno vidíme, že $\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP}$. Nevíme, jestli platí rovnost. Existují problémy (např. parity games, faktorizace čísel), které jsou v $\mathbf{NP} \cap \mathbf{coNP}$, ale zatím pro ně neznáme polynomický algoritmus.

2.5 Intermediate problémy: Lardnerova věta

Věta 15: Lardnerova

Pokud $\mathbf{P} \neq \mathbf{NP}$, pak existuje jazyk $L \in \mathbf{NP}$ tak, že $L \notin \mathbf{P}$ a současně L není **NP**-težký.

Důkaz. Diagonalizací. Zatím vynechán. □

Kandidáty na intermediate problémy jsou např. grafový izomorfismus (v současnosti je znám quasipolynomiální algoritmus) a faktorizace čísel.

Literatura

Definice tříd, verifikovatelnosti a redukcí lze nalézt ve všech knihách v referencích. Ukázky redukcí byli čerpány z [1] (kap. 7). Důkaz Cook-Levinovi věty, definice týkající se obvodů a diskuze **NP**vs **coNP**je zejména z [3](kap. 4, 8, 10). Podobně laděný důkaz lze nalézt i v [2](kap. 6).

3 Alternace

Alternující TS (ATS) je nedeterministický TS (s množinou stavů Q) obohacený o funkci $\sigma : Q \mapsto \{\vee, \wedge\}$ s následující definicí přijímání. O konfiguraci c, obsahující stav q, řekneme, že je *A-přijímající* pokud q není zamítací stav a dále:

a) je q přijímací stav

b) $\sigma(q) = \vee$ a existuje aspoň jedna konfigurace c', která je z c dosažitelná pomocí jednoho kroku, která je A-přijímající.

c) $\sigma(q) = \wedge$, a každá konfigurace c' dosažitelná z c pomocí jednoho kroku je A-přijímající.

ATS přijímá řetězec x, pokud je počáteční konfigurace pro x A-přijímající. Rychle vidíme, že nedeterministický TS je speciální typ ATS, kde jsou všechny stavy existenčně alternující.

Pokud $\sigma(q) = \vee$, říkáme, že q je existenčně alternující, jinak, pokud $\sigma(q) = \wedge$ je q univerzálně alternující. Obdobně o konfiguraci obsahující existenčně (univerzálně) alternující stav říkáme, že je existenčně (univerzálně) alternující.

Vztah ke hrám dvou hráčů. Pro jednoduchost předpokládejme, že ATS pro x vždy zastaví (tj. každá jeho výpočetní větev je konečná). Výpočet ATS pro x si můžeme představit jako konečnou hru, kterou hrají dva hráči p_0, p_1. Hra spočívá v pohybu po konfiguracích ATS. Ve hře máme vždy aktuální konfiguraci c, na začátku hry je konfigurací c počáteční konfigurace pro x. Tah hráče spočívá ve výběru konfigurace c' dosažitelné z c jedním krokem ATS, která bude novou aktuální konfigurací. Pokud je c existenciálně alternující, potom tah provádí hráč p_0. Je-li c univerzálně alternující, provádí tah hráč p_1. Hráč p_0 ve hře zvítězí v momentě, kdy je c je přijímací konfigurace, nebo hra skončí v univerzálně alternující konfiguraci, ze které neumí ATS provést další krok. Hráč p_1 vítězí v momentě, kdy c je zamítací konfigurace, nebo hra skončí v existenčně alternující konfiguraci, ze které neumí ATS provést další krok.

Platí, že ATS x přijímá x, právě když má p_0 vítěznou strategii (tu si prozatím představme intuitivně, podobně jako například v šachu, dámě, či jiné hře: p_0 umí vyhrát bez ohledu na to, jak táhne jeho soupeř. Přesnější definici uvidíme později v kapitole o třídě **PSPACE**).

Řekneme, že

- ATS M běží v čase $T(n)$, pokud jsou pro všechny x všechny výpočetní větve dlouhé nejvýše $T(|x|)$ kroků.
- ATS M běží s pamětí $S(n)$, pokud všechny x platí, že všechny výpočetní větve M během výpočtu navštíví nejvýše $S(|x|)$ různých políček pásky.

Zavedeme alternující složitostní třídy:

$$\mathbf{ATIME}(T(n)) = \{L(M) \mid M \text{ je ATS běžící v čase z } O(T(n))\}$$
$$\mathbf{ASPACE}(S(n)) = \{L(M) \mid M \text{ je ATS běžící pamětí z } O(S(n))\}$$

Věta 16: Vztah deterministických a alternujících tříd

Pro $t(n) \geq n$ a $s(n) \geq \log n$ platí

(i) **ATIME**$(t(n)) \subseteq$ **DSPACE**$(t(n))$

(ii) **DSPACE**$(s(n)) \subseteq$ **ATIME**$(s(n)^2)$

(iii) **ASPACE**$(s(n)) \subseteq$ **DTIME**$(2^{O(s(n))})$

(iv) **DTIME**$(t(n)) \subseteq$ **ASPACE**$(\log t(n))$

Důkaz. (i) Dokážeme, že k ATS s čas složitostí $f(n)$ najdeme TS s paměťovou složitostí v $O(f(n))$, který přijímá stejný jazyk. Upravíme důkaz Věty 1 (iv), kde dokazujeme **NTIME**$(t(n)) \subseteq$ **DSPACE**$(t(n))$. Připomeňme, že klíčová myšlenka v tomto důkazu je procházení stromu konfigurací pomocí rekurzivní procedury, přičemž si v rekurzivních zavoláních nepamatujeme aktuálních konfiguraci simulovaného NTS, ale globálně si pamatujeme posloupnost nedeterministických voleb, kterou se k dané konfiguraci NTS dostane. Velmi podobně procházíme strom konfigurací ATS i zde.

Změnou je to, že nyní počítáme (v aktuální zavolání rek. procedury), jestli je aktuální konfigurace A-přijímací (a máme tak vlastně navíc univerzální alternaci). To děláme snadnou úpravou: pokud je aktuální konfigurace univerzálně alternující, musí všechna rekurzivní zavolání pro konfigurace dosažitelné pomocí jednoho kroku z té aktuální, skončit úspěšně.

(ii) Ukážeme silnější tvrzení **NSPACE**$(s(n)) \subseteq$ **ATIME**$(s(n)^2$. Tedy pro NTS M s pamětovou složitostí $f(n)$ sestavíme ATS N se složitostí $O(f(n)^2)$.

Toto uděláme analogií důkazu Savitchovi věty (Věta 4). Jediná změna je v tom, že proceduru CAN-YIELD realizujeme pomocí alternujícího algoritmu, nikoliv pomocí deterministického algoritmu.

CAN-YIELD(α, β, m)

1. Pokud $m = 0$, pak přijímáme, pokud $\alpha = \beta$. Jinak zamítáme.
2. Pokud $m = 1$, pak přijímáme, pokud se NTS dostane z α do β pomocí jednoho kroku.
3. Jinak pomocí existenciální alternace (nedeterministicky) zvolíme konfiguraci γ.
4. Pomocí univerzální alternace poté provedeme rek. volání CAN-YIELD$(\alpha, \gamma, \lceil m/2 \rceil)$ a CAN-YIELD$(\gamma, \beta, \lfloor m/2 \rfloor)$.

Jedno volání procedury CAN-YIELD (do nějž nepočítáme rekurzivní volání) trvá $O(f(n))$ kroků, nedeterministicky v něm totiž volíme γ, jehož délka je omezena $O(f(n))$. Protože na začátku spouštíme CAN-YIELD s třetím argumentem rovným $d^{f(n)}$ (pro nějakou konstantu d), je hloubka stromu rekurze $O(f(n))$. Časová složitost N je tedy $O(f(n)^2)$.

(iii) K ATS M s pam. složitostí $f(n)$ sestavíme TS s čas složitostí $2^{Of(n)}$, který přijímá stejný jazyk. Uděláme to konstrukcí analogickou té z důkazu Věty 1 (v).

TS N (pro vstup x):

1. Na pásku vygeneruje všechny konfigurace stroje M, obsahující $f(n)$ políček pásky.

2. Opakovaně tyto konfigurace prochází a hledá A-přijímající konfigurace. Při prvním průchodu označí jako A-přijímající konfigurace, které obsahují přijímací stav, nebo univerzálně alternující konfigurace, ze kterých ATS neumí provést další krok a které neobsahují zamítací stav. V obecném průchodu označí za A-přijímající ty konfigurace, které jsou univerzálně alternující a všechny konfigurace dostupné z nich jedním krokem jsou již označeny jako A-přijímající; nebo konfigurace, které jsou existenčně alternující a aspoň jedna konfigurace dostupná z nich jedním krokem je již označena jako A-přijímající. Pokud již nenalezne žádnou konfiguraci, kterou lze označit, procházení končí.

3. Pokud je počáteční konfigurace (pro x) označena jako A-přijímající, N přijme. Jinak N zamítne.

N na pásku zapíše $2^{O(f(n))}$ konfigurací, jeden průchod konfiguracemi a označení nových A-přijímajících konfigurací zabere počet kroků, který je polynomický vzhledem k $2^{O(f(n))}$. Přitom je potřeba označit maximálně $2^{O(f(n))}$ konfigurací. Celkově je tedy složitost N nejvýše $2^{O(f(n))}$.

(iv) Předp. že máme TS M s čas složitostí $f(n)$. K němu můžeme (pro vstup x) sestavit tableau TAB (čtenář si konstrukci připomene z kapitoly 2.3).

Můžeme sestavit ATS N, který zkontroluje, že na konkrétním políčku tabulky je správný symbol. (Tj. symbol, který by se tam objevil, pokud by M počítal pro vstup x a postupně bychom tabulku doplňovali.) Pak stačí zkontrolovat, že ve druhém sloupečku na posledním řádku tabulky je správně symbol ACCEPT.

Zmiňovaný ATS N má jako vstup trojici (i, j, σ), kde i, j jsou indexy určující pozici v tabulce TAB, σ je symbol, jehož správné umístění na pozici $[i, j]$ kontrolujeme. Pro tento vstup N funguje následovně:

1. Pokud $i = 0$ (tedy jde o první řádek), pak ověříme správnost σ podle počáteční konfigurace M pro x.

2. Pokud $j = 0$ nebo $j = f(n)+1$ (tedy jde o první nebo poslední sloupec), pak ověříme jestli je σ ten symbol, který se v naší konstrukci v prvním či posledním řádků vyskytuje vždy.

3. Jinak nedeterministicky (pomocí existenciální alternace) vygenerujeme symboly $\sigma_1, \sigma_2, \sigma_3$, jako obsah políček TAB[i-1,j-1], TAB[i-1,j] a TAB[i-1,j+1]. Ověříme, že σ a $\sigma_1, \sigma_2, \sigma_3$ odpovídají přechodové funkci stroje M, pokud ne, pak zamítáme.

4. Pomocí univerzální alternace rekurzivně ověříme správnost pro trojice $(i-1, j-1, \sigma_1)$, $(i-1, j, \sigma_2)$, $(i-1, j+1, \sigma_3)$.

Vidíme, že N přijímá, právě když je obsah políčka správný (důkaz bychom provedli například indukcí přes řádky). Jeho paměťová složitost je $O(\log f(n))$, protože mu stačí si pamatovat konstantní počet indexů, které jsou v rozsahu 0 až $f(n)+1$ a zaberou tedy $O(\log f(n))$ políček pásky, a konstatní počet symbolů, z nich každý zabere konstantní počet políček pásky.

□

Z přechozího jsou pro nás důležité (pro příští kapitolu) zejména body 1 a 2, ze kterých plyne, že pokud definujeme

$$\mathbf{APTIME} = \bigcup_{k>0} \mathbf{ATIME}(n^k),$$
$$\mathbf{PSPACE} = \bigcup_{k>0} \mathbf{DSPACE}(n^k),$$

tak máme **APTIME** = **PSPACE**.

Literatura

Čerpáno z [2](kap 7).

4 Třída PSPACE

Nejdříve definujeme třídy

$$\mathbf{PSPACE} = \bigcup_{k>0} \mathbf{DSPACE}(n^k),$$
$$\mathbf{NPSPACE} = \bigcup_{k>0} \mathbf{NSPACE}(n^k),$$

Ze Savitchovi věty (Věta 4) hned plyne **PSPACE** = **NPSPACE**.

Pomocí redukce v polynomickém čase standardně definujeme pojmy **PSPACE**-těžký a **PSPACE**-úplný problém.

Plně kvantifikovaná booleovská formule je formule

$$(Q_1 x_1)(Q_2 x_2)\dots(Q_n x_n)\psi,$$

kde $Q_i \in \{\forall, \exists\}$ a ψ je výroková formule v CNF s proměnnými $x_1, \dots, x_n$. Kvantifikujme množinu $\{0, 1\}$. Pravdivost takové formule je definována obvyklým způsobem.

Definujeme jazyk

$$\mathsf{QBF} = \{\varphi \mid \varphi \text{ je pravdivá plně kvantifikovaná formule}\}$$

Speciální případy: pokud se omezíme na formule, ve kterých jsou všechny kvantifikátory existenciální, patří problém do **NP**. Pokud se naopak omezíme na formule, ve kterých jsou všechny kvantifikátory univerzální, patří problém do **coNP**.

Věta 17

$\mathsf{QBF} \in$ **PSPACE**

Důkaz. Alternující TM pro vstup $(Q_1 x_1)(Q_2 x_2)\dots(Q_n x_n)\psi$:

1. Alternací vytvoří ohodnocení proměnných $x_1, \dots, x_n$. Pokud je $Q_i = \forall$, ohodnotí x_i pomocí univerzální alternace, jinak, je-li $Q_i = \exists$, ohodnotí x_i pomocí existenciální alternace.
2. pro vytvořené ohodnocení spočítá pravdivost formule ψ.

Časová složitost je polynomická, protože proměnné můžeme ohodnotit v n krocích a spočítat pravdivost ψ lze v polynomickém čase vzhledem k velikosti ψ. Korektnost plyne z definice přijímání ATS. □

Věta 18

QBF je **PSPACE**-težký.

Důkaz. Uvažíme jazyk $L \in \mathbf{PSPACE}$. K němu existuje ATM M běžící v čase $m = n^k$ (pro nějakou konstantu k) tak, že $L = L(M)$.

Bez ztráty na obecnosti můžeme předpokládat, že pro každé x je

- počáteční konfigurace existenciálně alternující.
- z každé nekoncové konfigurace máme na výběr právě dvě konfigurace, do kterých můžeme přejít.
- v každé výpočetní větvi se pravidelně střídají univerzálně a existenciálně alternující konfigurace.

Předchozích vlastností lze dosáhnout úpravami, které mohou vyžadovat přidání konstantního počtu nových stavů.

Teď provedeme konstrukci z důkazu Cook-Levinovi věty, kdy pro vstup x vytvoříme formuli ψ nad proměnnými $y_1, \ldots, y_n$ (kde y_i odpovídá volbě výpočetní větve při provádění i-tého kroku výpočtu). Pokud nyní vytvoříme formuli

$$\varphi = (\exists y_1)(\forall y_2)(\exists y_3)(\forall y_4)\ldots(Q_m y_m)\psi,$$

ve které se pravidelně střídají kvantifikátory, pak díky předpokladům o M platí, že φ je pravdivá, právě když $x \in L$. Celá redukce je v polynomickém čase, protože ψ dokážeme vytvořit v polynomickém čase a doplnění kvantifikace také dokážeme doplnit v polynomickém čase, protože $|\varphi|$ a tedy i m jsou polynomické vzhledem k $|x|$. □

4.1 Vítězné strategie pro hry

Zobecněný slovní fotbal (angl. generalized geography) je hra hraná dvěma hráči na orientovaném grafu G, začínající v uzlu b.

Průběh hry: existuje jeden token, který je na začátku hry v uzlu b. Hráči se střídají v tazích (začíná hráč p_0). V jedno tahu hráč přesune token po hraně do sousedního, ve hře dosud nenavštíveného uzlu. Pokud takový uzel neexistuje, hráč, který je na tahu prohrává (a vítězí tak zbývající hráč). Všimneme si, že každá hra má nejvýše tolik tahů, kolik je vrcholů v grafu G.

Konfigurace hry je trojice (A, p, x), kde A je množina uzlů, na nichž už byl v průběhu hry token, $p \in \{p_0, p_1\}$ je hráč na tahu, x je uzel, na kterém je aktuálně token. Z konfigurace hra jedním tahem přejde do konfigurace (B, q, y) pokud $y \neq A$ je soused x, $B = A \cup \{x\}$ a $q = \bar{p}$ (kde $\overline{p_0} = p_1, \overline{p_1} = p_0$)), značíme $(A, p, x) \vdash (B, q, y)$.

Zavedeme (rekurzivně) predikát $\textsc{win}(p', (A, p, x))$, který je pravdivý pokud:

- $p' = p$ a existuje konfigurace (B, q, y) taková, že $(A, p, x) \vdash (B, q, y)$ a $\textsc{win}(p', (B, q, y))$ je pravdivý.
- $p' \neq p$ a pro každou konfiguraci (B, q, y) takovou, že $(A, p, x) \vdash (B, q, y)$ je predikát $\textsc{win}(p', (B, q, y))$ pravdivý. (Tento bod pokrývá i situaci, kdy p nemá žádný tah, protože univerzální kvantifikace přes prázdnou množinu je pravdivá z definice.)

Význam predikátu: $\textsc{win}(p', (A, p, x))$ je pravdivý, pokud má hráč p' z konfigurace (A, p, x) vítěznou strategii.

Zavedeme jazyk

$$GG = \{(G, b) \mid \textsc{win}(p_0, (\emptyset, p_0, b)) \text{ je pravdivý.}\}$$

Věta 19

GG je **PSPACE**-úplný.

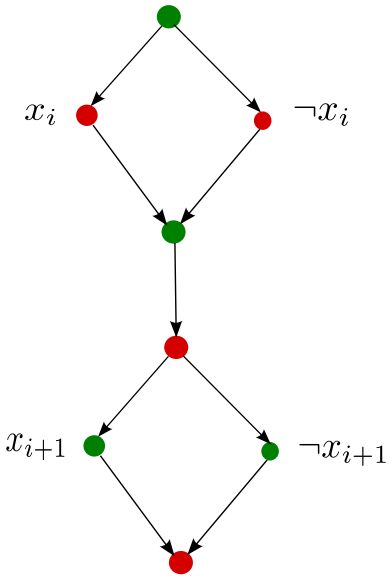
Důkaz. Redukcí z QBF. Uvažme tedy plně kvantifikovanou formuli

$$\varphi = (\exists x_1)(\forall x_2)(\exists x_3)\dots(\forall x_k)\psi.$$

Předpokládáme tedy, že se kvantifikátory pravidelně střídají a že k je dělitelné 2. To není předpoklad, který by ubral na obecnosti, protože každá plně kvantifikovaná formule jde rychle upravit do takového tvaru případným přidáním nadbytečných proměnných.

Vytvoříme graf G následovně. Pro každou dvojici ve tvaru $(\exists x_i)(\forall x_{i+1})$ vytvoříme podgraf (dvojici diamantů) tak, že během hry v tomto podgrafu hráč p_0 vybere ohodnocení proměnné x_i a hráč p_1 vybere ohodnocení proměnné x_{i+1}. Toto ohodnocení je definováno tak, že pokud se během hry dostane token do uzlu označeného literálem, ohodnotíme příslušnou proměnnou tak, aby byl tento literál nepravdivý.

Podgraf pro ohodnocení proměnných x_i a x_{i+1} je zachycen na následujícím obrázku. V zelených uzlech je na tahu hráč p_0 v červených uzlech hráč p_1. Hra začíná v horním zeleném uzlu.



Takové podgrafy (pro všechny dvojice kvantifikátorů) zřetězíme s pomocí přidaných hran za sebe tak, že hrou v nich hráči ohodnotí všechny proměnné. Označme poslední uzel tohoto řetězu diamantů v. První uzel tohoto

řetězu je uzel b (tj. uzel, kde hra začíná). Když je token v uzlu v, je tahu hráč p_1 a token byl ve všech uzlech mimo těch, které odpovídají pravdivým literálům.

Dále pro každou klausuli c ve formuli ψ vytvoříme jeden uzel. Do tohoto uzlu povede hrana z uzlu v, a z tohoto uzlu povede hrana do všech uzlů odpovídajících literálům obsažených v c.

Pokud není φ pravdivá, tak existuje klausule c, která je nepravdivá a tedy obsahuje pouze nepravdivé literály. Pokud hráč p_1 z uzlu v provede tah do uzlu klausule c, pak vyhraje. Z c totiž vede hrana pouze do uzlů nepravdivých literálů a ty už byly všechny navštíveny, takže p_0 nemůže z c přesunout token. Naopak, pokud je φ pravdivá, pak z každého uzlu odpovídajícího klausuli vede aspoň jedna hrana do doposud nenavštíveného uzlu (odpovídajícímu pravdivému literálu), takže hráč p_0 tam může přesunou token. Poté je tahu hráč p_1, ale nemá token kam přesunout, protože jediný soused aktuálního uzlu už byl navštíven při ohodnocování proměnných.

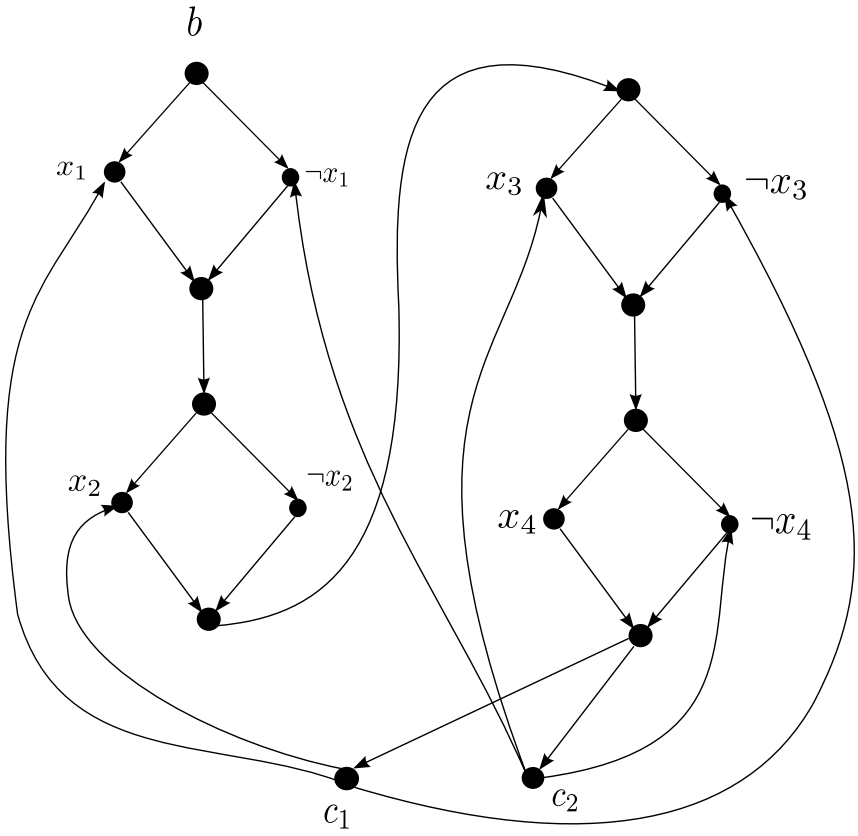
Protože hráč p_0 vybírá ohodnocení existenciálně kvantifikovaných proměnných a hráč p_1 vybírá ohodnocení univerzálně kvantifikovaných proměnných, je formule pravdivá právě když má hráč p_0 vítěznou strategii (to plyne přímo z definic).

Graf G můžeme očividně zkonstruovat v polynomickém čase (či dokonce v logaritmickém prostoru).

□

Na následujícím obrázku je instance GG, kterou získáme redukcí z předchozí věty z formule

$$(\exists x_1)(\forall x_2)(\exists x_3)(\forall x_4)(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4).$$



Příklady dalších **PSPACE**-úplných problémů

- Vítězná strategie v zobecněné dámě
- Jazyková ekvivalence nedeterministických automatů, regulárních výrazů, minimalizace nedeterministického automatu.

Literatura

Založeno na [2](kap. 8) a [1](kap. 8.2 a 8.3).

5 Polynomická hierarchie

Už jsme si ukázali, že CLIQUE $\in$ **NP**. Uvažme příbuzný problém

$$\text{EXACTCLIQUE} = \{(G, k) \mid \text{ největší klika v grafu } G \text{ má } k \text{ uzlů }\}.$$

Otázka, zda-li je EXACTCLIQUE $\in$ **NP** nás vede k úvahám o tom, jestli existuje krátký certifikát pro $(G, k) \in$ EXACTCLIQUE. U CLIQUE takový certifikát existuje, je to klika, která má alespoň k uzlů (stačí v pol. čase ověřit, že to je skutečně klika, která má k uzlů). Takový certifikát u EXACTCLIQUE nefunguje, protože bychom navíc museli ověřit, že každá klika v G má nejvýše k uzlů. Problém EXACT-CLIQUE se tedy zdá býti těžším než problémy z třídy **NP**. Tyto úvahy motivují následující definici.

Pro $k \geq 1$ řekneme, že jazyk L patří do třídy $\mathbf{\Sigma}_k^p$, pokud existuje TS M s polynomickou časovou složitostí a polynom q tak, že

$$x \in L \text{ právě když } (\exists u_1)(\forall u_2)(\exists u_3)\dots(Q_i u_k)M(x, u_1, u_2, \dots, u_k) = 1,$$

kde pro liché i máme $Q_i = \exists$ a pro sudé i máme $Q_i = \forall$, a pro $i = 1, \dots, k$ platí $|u_i| < p(|x|)$. Polynomická hierarchie je množina $\mathbf{PH} = \bigcup_{i \geq 1} \mathbf{\Sigma}_i^p$.

Vidíme, že EXACTCLIQUE patří do $\mathbf{\Sigma}_2^p$, a že $\mathbf{\Sigma}_1^p = \mathbf{NP}$. Z definice také hned vidíme, že $\mathbf{\Sigma}_i^p \subseteq \mathbf{\Sigma}_j^p$ pro $i \leq j$ (na konec můžeme přidat kvantifikátory, které stroj M ignoruje).

Nyní třídu $\mathbf{\Sigma}_k^p$ (pro $k \geq 1$) charakterizujeme pomocí (omezených) alternujících TS s polynomickou časovou složitostí. To je analogie charakterizace **NP** pomocí nedeterministických TS s polynomickou časovou složitostí.

Řekneme, že ATS M je $\mathbf{\Sigma}_k$-stroj, pokud pro každý jeho vstup a každou výpočetní větev platí

- výpočet můžeme rozdělit do maximálně k intervalů,
- v každém intervalu jsou jenom univerzálně nebo jenom existenciálně alternující konfigurace,
- první interval obsahuje existenciálně alternující konfigurace,

Věta 20

$\mathbf{\Sigma}_k^p = \{\, L(M) \mid M \text{ je } \mathbf{\Sigma}_k\text{-stroj s polynomickou časovou složitostí} \,\}$

*Důkaz. implikace zleva doprava.*Předp, že $L \in \mathbf{\Sigma}_k^p$. Vytvoříme ATS, který pro x nejdříve vytvoří $u_1, \dots, u_k$ a to pomocí existenciální alternace pro liché indexy, a univerzální alternace pro sudé indexy. Potom simuluje M pro vstup $x, u_1, \dots, u_k$. Protože délky $u_1, \dots, u_k$ jsou polynomicky omezeny a M pracuje v polynomickém čase, pracuje ATS také v polynomickém čase. Z definic snadno vidíme, že ATS je $\mathbf{\Sigma}_k$-stroj, a že přijímá, právě když $x \in L$.

implikace zprava doleva. Předp. že existuje polynomický $\mathbf{\Sigma}_k$-stroj N a $L = L(N)$. Fixujeme vstup x. Každou výpočetní větev stroje N lze rozdělit do nejvýše k intervalů, že se střídají intervaly s ex. a univ. alt. konfiguracemi. N běží v polynomickém čase, existuje tedy polynom q tak, že každý intervalů obsahuje nejvýše $q(|x|)$ konfigurací.

Nedeterministické volby, které určují konkrétní výpočetní větev lze tedy pro každý intervalu lze reprezentovat řetězcem délky maximálně $q(|x|)$.

Uvažme TS M, který pro vstup $x, u_1, u_2, \ldots, u_k$ simuluje činnost N pro x (jako NTS, alternace tu nehrají roli) a výpočetní větev přitom vybírá podle $u_1, u_2, \ldots$. Předp. že M přijímá pokud N zastaví a dostane se do A-přijímací konfigurace. Z definic hned plyne, že N přijímá x právě když

$$(\exists u_1)(\forall u_2)(\exists u_3)\ldots(Q_k u_k)\ \ M(x, u_1, u_2, \ldots, u_k) = 1,$$

a pro $i = 1, \ldots, k$ je $|u_i| < q(|x|)$. □

Duální ATS k ATS M (který vždy zastaví) dostaneme tak, že v M prohodíme alternace u stavů, a přijímací a zamítací stav. Jasně vidíme, že pro každý vstup mají M i k němu duální ATS stejné výpočetní stromy, ale změnily jsme přijímaný jazyk. Indukcí snadno dokážeme, že konfigurace ve výpočetním stromu M je A-přijímací, právě když tatáž konfigurace ve výpočetním stromu duálního ATS A-přijímací není. Tedy platí věta.

Věta 21

Nechť M je ATS, který vždy zastaví, a N je k němu duální ATS. Pak $L(M) = \overline{L(N)}$.

Zavedeme třídy $\mathbf{\Pi}_k^p = \mathbf{co\Sigma}_k^p$. Pokud definujeme $\boldsymbol{\pi}_k$-stroj analogicky $\boldsymbol{\Sigma}_k$-stroji pouze s tím, že první interval obsahuje univerzálně alternující konfigurace, dostaneme z předchozí věty hned

$$\mathbf{\Pi}_k^p = \{\, L(M) \mid M \text{ je } \boldsymbol{\pi}_k\text{-stroj s polynomickou časovou složitostí} \,\}.$$

Odtud analogií Věty 20 dostaneme $L \in \mathbf{\Pi}_k^p$ když existuje polynom q a TS M s pol. čas. slož tak, že

$$x \in L \text{ právě když } (\forall u_1)(\exists u_2)(\forall u_3)\ldots(Q_k u_k)M(x, u_1, u_2, \ldots, u_k) = 1,$$

kde pro liché i je $Q_i = \forall$ a pro sudé i je $Q_i = \exists$ a pro $i = 1, \ldots, k$ je $|u_i| \leq q(|x|)$.

Věta 22

(a) Pro $k \geq 1$ platí $\mathbf{\Sigma}_k^p \cup \mathbf{\Pi}_k^p \subseteq \mathbf{\Sigma}_{k+1}^p \cap \mathbf{\Pi}_{k+1}^p$

(b) **PH** $\subseteq$ **PSPACE**

Důkaz. (a) Tvrzení plyne z toho, že v definicích $\mathbf{\Sigma}_k^p$ a $\mathbf{\Pi}_k^p$ můžeme na konci (nebo začátku) přidat jeden kvantifikátor (a kvantifikovaný řetězec), který ovšem TS M ignoruje.

(b) Všechny jazyky z **PH** jsou přijímány ATS s konstantním počtem alternací. Pro problémy ve třídě **PSPACE** je ovšem počet alternací omezen polynmicky (viz Věta 16 a její důsledek: **APTIME** = **PSPACE**). □

Věta 23: O kolapsu polynomické hierarchie

(a) Pokud $\mathbf{P} = \mathbf{NP}$, pak $\mathbf{P} = \mathbf{PH}$

(b) Pro $i \geq 1$, pokud $\mathbf{\Sigma}_i^p = \mathbf{\Pi}_i^p$ pak $\mathbf{PH} = \mathbf{\Sigma}_i^p$.

Důkaz. (a) Pomocí indukce přes i dokážeme, že pro všechna i máme $\mathbf{\Sigma}_i^p, \mathbf{\Pi}_i^p \subseteq \mathbf{P}$. Pro $i = 1$ tvrzení triviálně plyne z předpokladu $\mathbf{P} = \mathbf{NP}$, protože $\mathbf{\Pi}_1^p = \mathbf{coNP}$ a $\mathbf{\Sigma}_1^p = \mathbf{NP}$.

Předpokládejme $i > 1$ a že tvrzení platí pro $i - 1$. Uvažme jazyk $L \in \mathbf{\Sigma}_i^p$. Z definice existuje polynom q a TS M s polynomickou časovou složitostí tak, že

$$x \in L \text{ právě když } (\exists u_1)(\forall u_2)(\exists u_3) \dots (Q_i u_i)\ M(x, u_1, u_2, \dots, u_i) = 1, \tag{3}$$

kde $|u_j| \leq q(|x|)$ pro $j = 1, \dots, i$. Definujeme jazyk

$$L' = \{\langle x, u_1 \rangle \mid (\forall u_2)(\exists u_3) \dots (Q_i u_i)\ M(x, u_1, u_2, \dots, u_i) = 1 \text{ a } |u_j| \leq q(|x|) \text{ pro } j = 1, \dots, i\}$$

Máme $L' \in \mathbf{\Pi}_{i-1}^p$ a podle indukčního předpokladu $L' \in \mathbf{P}$. To znamená, že existuje TS N s polynomickou časovou složitostí rozhodující L'. Dosazením zpět do (3) dostaneme

$$x \in L \text{ právě když } (\exists u_1)\ N(x, u_1) = 1,$$

kde $|u_1| \leq q(|x|)$. Tedy $L \in \mathbf{NP}$ a z předpokladu $\mathbf{P} = \mathbf{NP}$ pak plyne $L \in \mathbf{P}$.

(b) Analogicky jako (a). □

5.1 Úplné problémy pro jednotlivé úrovně hierarchie

Uvažujeme $\leq_p$ redukce. Definujeme jazyk

$$H_k = \{M\#x\#^m \mid M \text{ } (m, k)\text{-přijímá } x \text{ a jeho počáteční stav je existenčně alternující}\},$$

kde (m, k)-přijetí x je definováno následovně: Ve výpočetním stromu M pro x zkrátíme větve, na kterých je provedeno více než m kroků nebo obsahující více než k intervalů alternace (viz definice Σ_k nebo Π_k strojů) tak, aby měli nejvýše m kroků a k intervalů alternace (v jedné z možností nastane rovnost). Na tento strom aplikujeme běžnou definici A-přijímání.

Věta 24

H_k je $\mathbf{\Sigma}_k^p$-úplný.

Důkaz. Nejdříve ukážeme, že $H_k \in \mathbf{\Sigma}_k^p$. Vytvoříme $\mathbf{\Sigma}_k$-stroj N, který pro vstup y

1. deterministicky zkontroluje, že $y = M\#x\#^m$, kde M je popis ATS (jinak zamítne).

2. Simuluje M pro x, přičemž dělá univerzální (existenciální) alternace právě když M dělá univerzální (existenciální) alternace.

3. N si při simulaci počítá počet provedených kroků i počet změn alternací ve výpočtu M a pokud překročí limity (k nebo $m-1$) pak zastaví (a přijme nebo zamítne v závislosti na tom, v jak alternující konfiguraci skončí). Simulační overhead je deterministický a používá takovou alternaci, abychom nezvýšili počet použitých alternací.

Simulovat jeden krok lze v polynomickém čase (vzhledem k délce y) a simulujeme nejvýše m kroků. Protože $m \leq |y|$, je celý výpočet v polynomickém čase.

Nyní ukážeme, že H_k je $\mathbf{\Sigma}_k^p$-těžký. Uvažujeme $A \in \mathbf{\Sigma}_k^p$ a $\mathbf{\Sigma}_k$-stroj M jej s čas. složitostí omezenou polynomem q. Pokud pro vstup x nastavíme $m = q(|x|)$, a očividně platí

$$M \text{ přijímá } x \text{ právě když } M \text{ } (m,k)\text{-přijímá } x \text{ právě když } M\#x\#^m \in H_k.$$

Potřebná redukce je tedy zobrazení

$$x \mapsto M\#x\#^{q(x)},$$

které je snadno spočitatelné v polynomickém čase vzhledem k x. □

5.2 Oracle stroje a relativizované složitostní třídy

Už známe definici TS s přístupek k oracle pro jazyk L. Nyní budeme uvažovat takové TS, které mají navíc omezenu časovou složitost. Definujeme následující (relativizované) složitostní třídy. Pro jazyk B

$$\mathbf{P}^B = \{L(M) \mid M \text{ je TS s pol. čas. slož. a přístupem k oracle pro jazyk } B\}$$
$$\mathbf{NP}^B = \{L(M) \mid M \text{ je NTS s pol. čas. slož. a přístupem k oracle pro jazyk } B\}$$

Pro množinu jazyků $\mathcal{L}$

$$\mathbf{P}^{\mathcal{L}} = \bigcup_{L \in \mathcal{L}} \mathbf{P}^L$$
$$\mathbf{NP}^{\mathcal{L}} = \bigcup_{L \in \mathcal{L}} \mathbf{NP}^L$$

Připomeňme, že TS M s přístupem k oracle pro B chápeme jako Turingovskou redukci $L(M) \leq_T B$, teď navíc omezujeme její časovou složitost (a také u nedeterministických tříd specifikujeme, že může být nedeterministická). Očividně $L_1 \leq_p L_2$ implikuje $L_1 \leq_T L_2$, protože many-to-one redukce je speciální případ Turingovské redukce. Opačná implikace ovšem platit nemusí: máme například $SAT \leq_T \overline{SAT}$, ale $SAT \leq_p \overline{SAT}$ by implikovalo $\mathbf{NP} = \mathbf{coNP}$ (a to nevíme, jestli platí: myslíme si, že spíše ne).

Dále definujeme

$$\mathbf{NP}_1 = \mathbf{NP}$$
$$\mathbf{NP}_{k+1} = \mathbf{NP}^{\mathbf{NP}_k}$$

Věta 25

$\mathbf{NP}_k = \mathbf{\Sigma}_k^p$

Důkaz. Vynecháme. □

Literatura

[4](kap. 5), [2](kap. 9)

6 Třídy L a NL

Potřebujeme upravit definici TS (a NTS)[1]. Budeme předpokládat, že TS má k dispozici vstupní pásku, která je pouze pro čtení a je na ní zapsaný vstup a pracovní pásku, která je pro čtení a zápis a je na začátku výpočtu prázdná. K obsahuje čtecí pásky přistupuje pomocí indexů políček zapsaných v binární soustavě, délka indexu je tak v logaritmickém vztahu k délce vstupu. Pamětovou složitost měříme na pracovní pásce.

Definujeme třídy

$$\mathbf{L} = \mathbf{DSPACE}(\log n)$$
$$\mathbf{NL} = \mathbf{NSPACE}(\log n)$$

a problém

$$\mathsf{PATH} = \{(G, s, t) \mid G \text{ je orientovaný graf, ve kterém existuje cesta z uzlu } s \text{ do uzlu } t\}.$$

Věta 26

$\mathsf{PATH} \in \mathbf{NL}$.

Důkaz. Sestavíme NTS M, který potřebuje pouze logaritmickou pamět. Předp. že vstup je $(G = (V, E), s, t)$. M má na pracovní pásce zapsán aktuální uzel, na začátku tam zapíše s. Opakovaně nedeterministicky vybírá souseda aktuálního uzlu s a bude považuje jej za nový aktuální uzel. Současně si na pásce pamatuje počet výběrů nového uzlu, které provedl. Pokud má na pracovní pásce t, pak přijímá. Jinak, pokud již provedl tolik výběrů, kolik je uzlů v grafu, zamítá. M reprezentuje uzly jako čísla zapsaná v binární soustavě, k reprezentaci jednoho čísla mu stačí $\lceil \log_2 |V| \rceil$ bitů. Stejně tak k uchování počtu provedených voleb mu stačí logaritmická pamět. (Další pamět M může potřebovat k pamatování si indexů políček na čtecí pásce, ty ovšem také reprezentuje v binární soustavě a stačí mu tak logaritmická pamět). □

Nevíme, jestli $\mathsf{PATH} \in \mathbf{L}$ (zajímavé je, že verze tohoto problému pro neorientované grafy je v $\mathbf{L}$). Obecně nevíme, jestli $\mathbf{L} = \mathbf{NL}$ (víme jenom $\mathbf{L} \subseteq \mathbf{NL}$).

Log-space transducer je TS pracující v logaritmickém prostoru (jak jsme jej diskutovali na začátku kapitoly), který je navíc vybaven výstupní páskou, která je pouze pro zápis. Poté, co zapíše na políčko této pásky symbol, posune zapisovací hlavu doprava. Přitom ji nikdy nemůže posunout doleva (tj. co je jednou zapsáno už nejde přepsat). Samotný zápis může být realizován například přechodem do speciálního zapisovacího stavu atd.

Funkce $f : \Sigma^* \mapsto \Sigma^*$ je vyčíslitelná v logaritmickém prostoru, pokud existuje log-space transducer, který pro vstup x zastaví a na výstupní pásce zůstane $f(x)$.

Problém A je redukovatelný v logaritmickém prostoru na problém B, značeno $A \leq_{\log} B$, pokud existuje funkce f vyčíslitelná v logaritmickém prostoru tak, že $x \in A$ právě když $f(x) \in B$.

Snadno vidíme, že pokud $A \leq_{\log} B$ pak $A \leq_p B$. V současnosti není známo, zda platí i opačná implikace, i když všechny polynomické redukce, které jsme si doposud ukázali, jsou vlastně redukce v logaritmickém prostoru.

[1] Všechny důkazy, které jsme dosud dělali, by šli snadno upravit na tento nový model

Věta 27

$A \leq_{\log} B$ a $B \in L$ implikuje $A \in \mathbf{L}$.

Důkaz. Protože $B \in L$ existuje TS M, který rozhoduje B v log. prostoru. Protože $A \leq_{\log} B$ existuje log-space transducer T počítající funkci f_T odpovídající redukci.

Sestavíme TS N, který rozhoduje A. Pro vstup x:

1. Simuluje M. Vždycky, když M čte i-tý symbol ze vstupní pásky, N simuluje od začátku výpočet T pro vstup x, dokud T nezapíše na výstupní pásku i-tý symbol. Tento symbol poté použije jako i-tý vstupní symbol pro M.
2. Pokud M přijímá, N také přijímá. Jinak N zamítá.

N potřebuje pouze logaritmickou pamět:

- Potřebuje pamět pro běh M. M potřebuje $O(\log |f_T(x)|)$ paměti, ale protože $|f_T(x)| \leq |x|^k$ (pro nějakou konstantu k) a $\log x^k = k \cdot \log x$, je velikost potřebné paměti logaritmická i vzhledem k $|x|$.
- Potřebuje pamět pro běh T. Ta je logaritmická vzhledem k x. N si také nemusí pamatovat celý výstup T, stačí si pamatovat kolik symbolů už T vytiskl (to je polynomicky omezené číslo, které zabere zapsáno v binární abecedě logaritmický prostor) a poslední symbol, který T vytiskl.

□

Věta 28

$A \leq_{\log} B$ a $B \leq_{\log} C$ implikuje $A \leq_{\log} C$.

Důkaz. Potřebný transducer sestavíme analogicky sestavení TS N v důkazu předchozí věty. Trasducer realizující $A \leq_{\log} B$ hraje roli T a transducer realizující $B \leq_{\log} C$ hraje roli M. □

Jazyk A je **NL**-těžký, pokud pro každý $B \in \mathbf{NL}$ máme $B \leq_{\log} A$. Pokud je navíc $A \in \mathbf{NL}$, pak je A **NL**-úplný. V důsledku předchozí věty pak platí, že pokud je A **NL**-těžký a $A \leq_{\log} B$, pak je i B **NL**-těžký.

Věta 29

PATH je **NL**-těžký problém.

Důkaz. Předp. že $A \in \mathbf{NL}$ a M je NTS rozhodující A v log. pam. složitostí, řekněme $s(n) \in O(\log(n))$. Dále bez ztráty na obecnosti předpokládejme, že M má unikátní přijímací konfiguraci (např. smaže obsah pásky a přemístí hlavu na začátek), tuto konfiguraci označíme ACCEPT.

Pro vstup x uvážíme orientovaný graf $G = (V, E)$, V je tvořena všemi konfiguracemi stroje M, které obsahují $s(|x|)$ políček pásky, a

$$E = \{(c_1, c_2) \mid \text{z konfigurace } c_1 \text{ přejde } M \text{ do konfigurace } c_2 \text{ pomocí jednoho kroku}\}.$$

Jasně vidíme, že $x \in L$ právě když v G existuje orientovaná cesta z počáteční konfigurace do konfigurace ACCEPT.

Zbývá říci, jak můžeme G vytvořit na základě x pomocí transduceru s log. pam. slož. Potřebný transducer T pro vstup x:

1. zapíše na výstupní pásku v lexikografickém uspořádání všechny konfigurace stroje M s $s(|x|)$ políčky pásky. K tomu mu stačí $O(\log |x|)$ paměti.

2. zapíše na pracovní pásku postupně všechny dvojice konfigurací TS M s $s(|x|)$ políčky pásky (na pásce je vždy zapsána jedna taková dvojice). Pro každou takovou dvojici ověří, jestli se M dostane z první konfigurace do té druhé pomocí jednoho kroku. Pokud ano, zapíše transducer danou dvojici na výstupní pásku. K provedení tohoto kroku je opět potřeba pouze $O(\log |x|)$ políček pásky.

3. nakonec zapíše na výstupní pásku počáteční konfiguraci a konfiguraci ACCEPT.

□

Důsledkem předchozí věty je například **NL** $\subseteq$ **P**, protože PATH $\in$ **P**. Immermanovi-Szelepcsenyiho věta implikuje (**NL** = **coNL**).

Literatura

Čerpáno z [1](kap. 8) a [2](kap. 5).

7 Paralelní složitost

Rodina booleovských obvodů $\mathcal{C}$ je posloupnost booleovských obvodů $C_0, C_1, C_2 \ldots$, kde obvod C_n má n vstupních bran. Řekneme, že $\mathcal{C}$ rozhoduje jazyk $A \subseteq \{0,1\}^*$ pokud pro každý řetězec $x \in \{0,1\}^*$ platí $x \in A$ právě když $C_{|x|}(x) = 1$.

Hloubka booleovského obvodu C je délka nejdelší cesty ze vstupní brány do výstupní brány. Velikost C je počet bran, které obsahuje.

Rodina booleovských obvodů $\mathcal{C}$ je uniformní, pokud existuje log-space transducer, který pro vstup 0^n zapíše na výstupní pásku popis obvodu C_n.

Booleovský obvod pro násobení binárních matic Máme matice A a B o rozměrech $n \ldots n$. Výstupem je matice C o rozměrech $n \times n$, která je booleovským součinem A a B. (tedy $C_{ij} = \bigvee_{k=1}^{n} A_{ik} B_{kj}$). Cílený obvod má $2n^2$ vstupních bran označených a_{ij} a b_{ij} (pro $1 \leq i, j \leq m$) korespondujících s prvky A_{ij},B_{ij} jednotlivých matic. Dále má n^3 bran g_{ijk} (pro $1 \leq i, j, k \leq n$), přičemž brána g_{ijk} počítá $a_{ik} \wedge b_{kj} b$. Pro každou dvojici i, j potom realizujeme pomocí obvodu formuli $\bigvee_{1 \leq k \leq n} g_{ijk}$. To lze provést obvodem s max $2n$ branami a hloubkou $O(\log n)$, přičemž poslední bránu v tomto obvodu ponecháme jako výstupní (a označíme ji c_{ij}). Dostaneme tak tedy obvod s nejvýše $2n^2 + n^3 + 2n^3$ branami a hloubkou $O(\log n)$.

Booleovský obvod pro tranzitivní uzávěr Pro výpočet tranzitivní uzávěru Booleovské matice A o velikosti $n \times n$ stačí spočítat mocniny A^i pro $i = 2, \ldots, n$ a spočítat $\bigvee_{i=1}^{n} A^i$. To můžeme provést pomocí opakovaného umocňování, kdy sérií $O(\log m)$ součinů matic spočítáme A^2, A^4,A^8, ... Poté vždy pro pomocí maximálně $O(\log n)$ dalších součinů dopočítáme A^i pro i, která nejsou mocninou dvou (stačí si představit i zapsáno v binární soustavě). Pro všechna taková i to ovšem provedeme paralelně.

Pro násobení matic máme z předchozího příkladu polynomicky velký booleoský obvod s hloubkou $O(\log n)$ a polynomickou velikostí. Protože počítáme $O(\log n)$ součinů za sebou, získáme obvod s hloubkou $O(\log^2 n)$. Tento obvod má také polynomicky mnoho bran. Připomeňme vztah problému dosažitelnosti v (orientovaných) grafech: pokud vezmeme matici sousednosti grafu a spočítáme její tranzitivní uzávěr, pak ve výsledku ja na pozici odpovídající uzlům i, j jednička, právě když existuje cesta z uzlu i do uzlu j.

Pro $i \geq 1$ řekneme, že $L \in \mathbf{NC}_i$ když existuje uniformní rodina booleovských obvodů $\mathcal{C}$ rozhodující L taková, že velikost obvodu C_n je $n^{O(1)}$ a hloubka C_n je $O(\log^i n)$

Dále definujeme

$$\mathbf{NC} = \bigcup_{i \geq 1} \mathbf{NC}_i.$$

Věta 30

(a) $\mathbf{NC}_1 \subseteq L$

(b) $\mathbf{NL} \subseteq \mathbf{NC}_2$

(c) $\mathbf{NC} \subseteq \mathbf{P}$

Důkaz. Bez újmy na obecnosti předpokládejme, že pracujeme na abecedou $\{0,1\}$.

(a) Z předpokladů víme, že pro $A \in \mathbf{NC}_1$ existuje uniformí rodina obvodů s $O(\log n)$ hloubkou a polynomickou velikostí, který rozhoduje A. Navíc existuje log-space transducer T, který pro vstup 0^n vytvoří obvod pro požadovanou velikost vstupu (tedy $C_{|x|}$ pro vstup x).

Vytvoříme TS M s log. pam. složitostí, který také rozhoduje A. Předpokládejme tedy vstup $x \in \{0,1\}^n$ Důležité je si uvědomit, že M může opakovaně simulovat T, a získat tak část obvodu C_n, kterou zrovna potřebuje. Poznamenejme, že M pro simulaci T nemusí nikam zapisovat řetězec 0^n, tedy vstup pro T, stačí si pouze pamatovat, na které pozici vstupní pásky je čtecí hlava. K tomu stačí čítač, k jehož uložení stačí $O(\log n)$ paměti.

TS M provádí průchod do hloubky grafem, který dostaneme z obvodu C_n tak, že obrátíme směr hran. Průchod začíná výstupní bránou. Přitom (očividným způsobem) počítá hodnoty jednotlivých bran, a nakonec i té výstupní, podle její hodnoty přijímá či zamítá. (Tento průchod do hloubky si můžeme také představit jako rekurzivní proceduru výpočtu hodnoty brány). TS M si v průběhu výpočtu (na pracovní pásce) pamatuje

- aktuální bránu,
- řetězec jednoznačně určující větev průchodu do hloubky, které odpovídá aktuální brána,
- mezivýsledky pro úrovně průchodu, ve kterých jsme do hloubky prošli jenom jeden podstrom, ale brána má aritu 2.

Protože C_n má polynomicky mnoho bran stačí k reprezentaci jedné logaritmický počet bitů. Protože brány mají aritu max 2, a hloubka C_n je logaritmická, pro uložení řetězce určujícího větev průchodu stačí logaritmický počet bitů. Díky logaritmické hloubce C_n nám také pro ukládání mezivýsledků stačí logaritmický počet bitů.

(b) Uvažujeme $A \in \mathbf{NL}$ a NTS M s log. pam. slož. tak, že $L(M) = A$. (O M bez ztráty na obecnosti předpokládáme, že má unikátní přijímací konfiguraci ACCEPT). Uvažujeme vstup $x \in \{0,1\}^n$. Ukážeme, že lze (log-space transducerem) zkonstruovat obvod C_n tak, že $x \in A$ právě když $C_n(x) = 1$.

Všimneme si, že konfigurace stroje M neobsahují vstupní pásku (ale pouze index toho, na jakém pozici se vyskytuje čtecí hlava na vstupní pásce) a tedy množina konfigurací M nezávisí na x, ale pouze na n.

Pokud bychom ovšem chtěli sestavit graf konfigurací M (tj. graf z důkazu věty 28), už x potřebujeme. Pokud je v konfiguraci α hlava na vstupní pásce na pozici i, závisí množina konfigurací dosažitelných jedním krokem z α na i-tém symbolu x.

Na základě M tedy můžeme vytvořit obvod C se vstupními branami $x_1, \ldots, x_n$ (ty odpovídají jednotlivým symbolům v x) a dále s branami

$$g_{ij} \text{ pro } 0 \leq i, j < n$$

tak, že pokud interpretujeme hodnotu brány g_{ij} (pro ohodnocení dané $x = x_1 x_2 \ldots x_n$) jako hodnotu B_{ij} v matici B, pak je B maticí grafu konfigurací TS M pro vstup x. Výsledný obvod má (vzhledem k n) polynomický počet bran a konstantní hloubku.

Protože je velikost jedné konfigurace M logaritmická (vzhledem k n), a jejich počet je tedy omezen polynomicky, můžeme C zkonstruovat log-space transducerem (jehož vstupem je 0^n)

Nyní si uvědomíme, že tímtéž log-space transducerem můžeme C rozšířit (na obvod C_n) tak, že za něj připojíme obvod pro výpočet tranzitivního uzávěru matice o dimenzích $n \times n$ tak, v něm považujeme g_{ij} za vstupní brány. Tento obvod má opět polynomický počet bran (vzhledem k n) a hloubku $O(\log^2 n)$.

Připomeňme, že výsledkem tranzitivního uzávěru matice sousednosti grafu, je matice dosažitelnosti (na pozici i, j je 1, právě když existuje cesta z vrcholu i do vrcholu j). Za výstupní bránu obvodu C_n označíme pozici v matici, která odpovídá dosažitelnosti konfigurace ACCEPT z počáteční konfigurace. Zjevně $x \in A$, právě když $C_n(x) = 1$.

(c) Předp. $A \in NC$. Pro vstup $x \in \{0,1\}^n$

1. pomocí log-space transduceru sestavíme C_n.
2. vypočítáme $C_n(x)$ a podle výsledku přijímáme.

Protože má C_n polynomický počet bran, lze druhý bod provést v polynomickém čase (obojí vzhledem k n). □

Problémy v **NC** považujeme za efektivně paralelizovatelné. Zajímavá otázka je, jestli platí $\mathbf{P} \subseteq \mathbf{NC}$, tedy zda-li jsou všechny problémy, pro které existuje efektivní sériový algoritmus také efektivně paralelizovatelné.

Jazyk L je **P**-težký pokud pro každý jazyk $L' \in \mathbf{P}$ platí $L' \leq_{\log} L$ a **P**-úplný, pokud je navíc v **P**.

Věta 31

$A \leq_{\log} B$ a $B \in NC$ implikuje $A \in NC$.

Kostra důkazu. (Opět bez újmy na obecnosti předpokládáme abecedu {0,1}). Pro obecnou velikost vstupu n dokážeme, že existuje obvod C_n tak, že pro $x \in \{0,1\}^n$ máme: $x \in A$ právě když $C_n(x) = 1$.

Nechť R je log-space transducer realizující redukci $A \leq_{\log} B$. Pak existuje polynom q tak, že R zapíše na výstup max $q(n)$ bitů. Dále určite existuje TS M s log. pam. slož., který pro vstup (x, i) přijímá, pokud je i-tý bit $R(x)$ roven 1. (Stačí simulovat R a počkat, až vypíše i-tý bit). Činnost M můžeme realizovat pomocí obvodu (viz důkaz předchozí věty), a tedy paralelním zapojením $q(|x|)$ obvodů získáme celý $R(x)$. Na něj nyní napojíme booleovský obvod pro B správné velikosti (ten získáme simulací log-space transduceru, který generuje obvody pro B). Získáme tak C_n.

Výše popsaný postup lze provést pomocí log-space transduceru.

□

Projdeme-li si pasáž o tableau metodě nad Větou 13 a uvědomíme si, že popsanou konstrukci lze provést pomocí log-space transduceru, dostáváme, že problém CIRCUIT-VALUE je **P**-úplný.

Hornova formule je formule výrokové logiky v CNF taková, že v každé klausuli je nejvýše jeden pozitivní literál.

$$\text{HORN-SAT} = \{\varphi \mid \varphi \text{ je splnitelná Hornova formule }\}$$

Věta 32

HORN-SAT je **P**-úplný.

Důkaz. Polynomický algoritmus pro vstup φ, opakuje následující:

1. Pokud φ neobsahuje klausuli s jedním literálem, můžeme ohodnotit všechny proměnné na 0. Tím ji splníme.
2. Pokud φ obsahuje klausule (x) a $(\neg x)$, pro nějakou proměnnou x, potom zamítáme.
3. Pokud φ obsahuje klausuli (l) s jedním literálem l, potom ohodnotí přislušnou proměnnou tak, aby byl l pravdivý. Formuli φ upraví tak, že odstraní klasuli (l) a ze zbývajících klausulí literál $(\neg l)$.

Algoritmus je korektní: klíčem je pozorování, že pokud nastane situace v bodu 3, pak je φ před úpravou splnitelná právě když je splnitelná formule, kterou dostaneme úpravou φ.

Algoritmus pracuje v polynomickém čase: v každé iteraci se formule zmenší (ubude jedna proměnná), může tedy být maximálně tolik iterací, kolik je proměnných.

Abychom ukázali, že HORN-SAT je P-težký, ukážeme CIRCUIT-VALUE$\leq_{\log}$HORN-SAT. Uvažujme tedy obvod C (bez proměnných). K němu vytvoříme formuli φ v CNF obsahující pouze hornovy klausule. Pro každou bránu g v C budeme mít dvě proměnné: g^+ a g^- (význam, který budeme chtít je, že g^+ je pravdivá, když brána g je pravdivá a g^- je pravdivá, když brána g není pravdivá). Pro každou takovou dvojici proměnných přidáme k φ klausuli $(\neg g^+ \vee \neg g^-)$, vylučující, aby byli obě proměnné současně pravdivé. Pro každou bránu v obvodu C pak přidáme k φ konjunkci klausulí.

- g konstantní brána: pokud je label g 1, vytvoříme klausuli g^+, jinak klausuli g^-
- g není konstantní brána a vedou do ní hrany z bran h_1, h_2 (pokud je label g negace, pak jenom hrana z h_1): vyvoříme hornovskou formuli vyjadřující, že pravdivost g je správně spočtena z pravdivosti h_1, h_2. Například, pokud má g label $\vee$, pak vytvoříme formuli

  $$(g^+ \vee \neg h_1^+) \wedge (g^+ \vee \neg h_2^+) \wedge (g^- \vee \neg h_1^- \vee \neg h_2^-)$$

  Vysvětlení: $(g^+ \vee \neg h_1^+)$ je ekvivalentní $h_1^+ \Rightarrow g^+$; $(g^+ \vee \neg h_1^+)$ je ekvivalentní $h_2^+ \Rightarrow g^+$; a konečně $(g^- \vee \neg h_1^- \vee \neg h_2^-)$ je ekvivalentní $(h_1^- \wedge h_2^-) \Rightarrow g^-$. (formule pro zbylé labely si laskavý čtenář sestaví jako cvičení).

Pro výstupní bránu g navíc k formuli přidáme klausuli g^+.

φ můžeme vytvořit pomocí log-space transduceru (máme vzory pro produkované části formule, ve kterých měníme jenom logaritmicky dlouhá zakódovaní proměnných).

To že je φ pravdivá, právě když je C pravdivý snadno dokážeme pomocí indukce přes brány uvažovány v pořadí daném nějakým jejich topologickým uspořádáním.

□

Třída AC Pokud definici Booleovského obvodu povolíme, aby brány s labely $\wedge, \vee$ měli neomezenou aritu a nikoliv pouze aritu 2, pak analogií třídy $\mathbf{NC}_i$ s takto pozměněnou definicí booleovského obvodu je třída $\mathbf{AC}_i$. Z definic hned plyne $\mathbf{NC}_i \subseteq \mathbf{AC}_i$. Bránu s aritou k můžeme simulovat obvodem používajícím pouze brány s aritou 2. Tento obvod má hloubku $O(\log k)$ a obsahuje $O(k)$ bran. Protože obvody pro problémy z $\mathbf{AC}_i$ obsahují pouze polynomický počet bran a arita všech bran je tedy omezena polynomicky, dostáváme že nahrazením všech bran s aritou různou ode dvou příslušným obvodem zvýšíme hloubku obvodu nanejvýš $O(\log n)$ krát. Odtud máme $\mathbf{AC}_i \subseteq \mathbf{NC}_{i+1}$. Celkem tedy

$$\mathbf{AC} = \bigcup_{i \geq 0} \mathbf{AC}_i = \bigcup_{i \geq 0} \mathbf{NC}_i = \mathbf{NC}.$$

Literatura

[2](kap. 11), [1](kap. 10.5), [5](str. 224).

8 Pravděpodobnostní algoritmy a příslušné třídy složitosti

Definice pravděpodobnostního TS (PTS) Uvažme NTS M takový, že z každé (nekoncové) konfigurace máme právě dvě nedeterministické volby. Představme si, že výpočet M pro x (tj. procházíme jednou větví jeho stromu výpočtu pro vstup x) je realizován následovně: Z konfigurace c

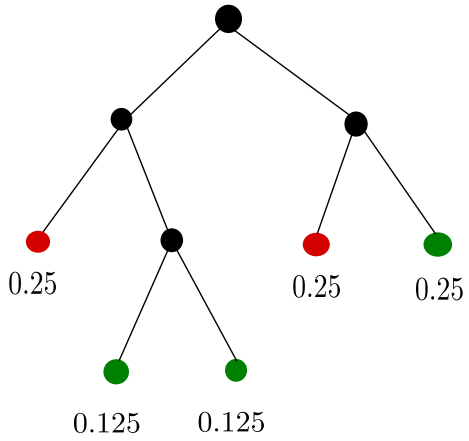
1. M si hodí mincí (s pstí 0.5 padne orel, s pstí 0.5 padne panna),
2. pokud padne orel, provede se výpočetní krok odpovídající první nedeterministické volbě přístupné z c, jinak krok odpovídající druhé volbě.

Předpokládejme, že M vždy zastaví. Větev v výpočtu, jejíž koncová konfigurace je v hloubce $h(v)$ je pak projita s pravděpodobností $1/2^{h(v)}$. Řekneme tedy, že M přijímá přijímá x s pravděpodobností

$$P[M(x) = 1] = \sum_{v \in S} 1/2^{h(v)},$$

kde S je množina přijímacích větví M pro vstup x. Očividně M zamítá s pstí $P[M(x) = 0] = 1 - P[M(x) = 1]$.

Složitost je u PTS definována jako u NTS (jde nám o délky větví etc.)



Obrázek 1: Příklad výpočetního stromu PTS s vyznačenými pravděpodobnostmi dosažení koncových konfigurací. Pravděpodobnost přijetí (zelené konfigurace) je 0.5, pravděpodobnost zamítnutí (červené konfigurace) je také 0.5

Třídy složitosti Řekneme, že $L \in$ **RP** pokud existuje PTS M s pol. čas. složitostí tak, že

$$x \in A \text{ implikuje } P[M(x) = 1] \geq 3/4$$
$$x \notin A \text{ implikuje } P[M(x) = 1] = 0.$$

PTS M tedy může udělat pouze jednostrannou chybu. Pokud přijímá, máme jistotu, že $x \in A$. Pokud zamítá, mohlo být (s pstí omezenou $1/4$) $x \in A$. Snadno vidíme, že ve třídě **coRP** je chyba přesunuta na druhou stranu.

Řekneme, že $L \in \mathbf{BPP}$, pokud existuje PTS M s pol. čas. slož. tak, že

$$x \in A \text{ implikuje } P[M(x) = 1] \geq 3/4$$
$$x \notin A \text{ implikuje } P[M(x) = 1] \leq 1/4.$$

PTS M tedy dělá chyby na obě strany (ale opět pstí omezenou $1/4$).

Věta 33: Amplifikace

Pokud $A \in \mathbf{BPP}$ potom pro libovolný polynom n^d existuje PTS s polynomickou časovou složitostí tak, že pro x délky n platí

$$P[M(x) \neq A(x)] \leq 1/2^{n^d}.$$

Důkaz. Díky $A \in \mathbf{BPP}$ existuje PTS M s pol. čas. slož rozhodující A s chybou omezenou $1/4$. Sestavíme PTS N, který simuluje M pro vstup x opakovaně, a to n^{d+1}-krát. N přitom spočítá četnost přijetí a zamítnutí v těchto výpočtech a přijímá, pokud ve více než polovině případů M přijímal. Protože složitost M je polynomická, je i složitost N polynomická. Pst, že N udělá chybu (tj. pst, že více než polovina simulací skončila s chybou) můžeme omezit

$$\sum_{k=0}^{n^{d+1}/2} \binom{n^{d+1}}{k} \cdot \left(\frac{3}{4}\right)^k \cdot \left(\frac{1}{4}\right)^{n^{d+1}-k} \leq \frac{1}{2^{n^d}},$$

□

Vztahy mezi třídami Snadno vidíme, že $\mathbf{P} \subseteq \mathbf{RP} \subseteq \mathbf{NP}$ a také $\mathbf{RP} \subseteq \mathbf{BPP}$ a $\mathbf{coRP} \subseteq \mathbf{BPP}$. Z definice **BPP** také plyne, že je uzavřená na doplněk.

Věta 34

$\mathbf{BPP} \subseteq \mathbf{\Sigma}_2^p$

Předtím, než si větu dokážeme, si uvědomíme, že na PTS se můžeme dívat následovně: uvažujeme obyčený TS M, kterému dáme navíc k dispozici jako další vstup dostatečně dlouhý řetězec bitů (umístěný na speciální pásce), který nahrazuje házení mincí ve výpočtu. Pro konkrétní vstup x a (dostatečně dlouhý) bitový řetězec y je pak výpočet $M(x, y)$ deterministický. Pst. že M se složitostí $T(n)$ přijímá x potom definujeme jako

$$P_y[M(x, y) = 1] = \frac{|\{y \in \{0,1\}^k \mid M(x, y) = 1\}|}{2^k},$$

kde k je libovolné číslo větší než $T(|x|)$.

Důkaz Věty 34. Uvažme $A \in \mathbf{BPP}$. Z Věty 33 plyne, že ex. TS s čas. slož n^c (pro nějakou konstantu $c > 1$) tak, že

- pokud $x \in A$, pak $P_y[M(x,y) = 1] \geq 1 - 1/2^n$ a
- pokud $x \notin A$, pak $P_y[M(x,y) = 1] \leq 1/2^n$.

Zafixujeme vstup x délky n a uvážíme $m = n^c$ (to je počet náhodných bitů, které M potřebuje). Definujeme množiny

$$A_x = \{y \in \{0,1\}^m \mid M(x,y) = 1\}$$
$$R_x = \{y \in \{0,1\}^m \mid M(x,y) = 0\}$$

Hned vidíme, že $A_x \cap R_x = \emptyset$ a $A_x \cup R_x = \{0,1\}^k$. Také pokud $x \in A$, tak $|A_x| \geq (1 - 1/2^n) \cdot 2^m = 2^m - 2^{m-n}$ a $|R_x| \leq 1/2^n \cdot 2^m = 2^{m-n}$. Pokud $x \notin A$, pak $|A_x| \leq 2^{m-n}$ a $|R_x| \geq 2^m - 2^{m-n}$.

Pomocí $\oplus$ budeme značit bitovou operaci XOR prováděnou na řetězcích z $\{0,1\}^m$. Připomeňme její základní vlastnosti, které využijeme. Pro každé $a, b, c \in \{0,1\}^m$ platí

$$a \oplus b = b \oplus a \tag{4}$$
$$(a \oplus b) \oplus c = a \oplus (b \oplus c) \tag{5}$$
$$a \oplus a = 0 \tag{6}$$
$$0 \oplus a = a \tag{7}$$
$$a \oplus b = c \text{ právě když } a = b \oplus c \tag{8}$$
$$a = b \text{ právě když } a \oplus c = b \oplus c \tag{9}$$

Operaci $\oplus$ nyní rozšíříme i na množiny. Pro množinu $S, S' \subset \{0,1\}^m$ a $a \in \{0,1\}^m$ máme

$$a \oplus S = \{a \oplus b \mid b \in S\},$$
$$S' \oplus S = \{a \oplus b \mid a \in S', b \in S\}$$

Vidíme, že v důsledku (9) máme $|a \oplus S| = |S|$, v důsledku (8) máme $a \oplus b \in S$ právě když $a \in b \oplus S$. Nyní dokážeme pomocné tvrzení kombinatorické povahy.

Tvrzení. $x \in A$ *právě když existují* $z_1, \ldots, z_m \in \{0,1\}^m$ *tak, že* $A_x \oplus \{z_1, \ldots, z_m\} = \{0,1\}^m$.

Důkaz pomocného tvrzení. Nejdřív předpokládejme, že $x \notin A$ a tedy $|A_x| \leq 2^{m-n}$. Pak pro jakékoliv $z_1, \ldots, z_m$ máme

$$A_x \oplus \{z_1, \ldots, z_m\} \leq \bigcup_{j=1}^{m} z_j \oplus A_x$$

a velikost této množiny tak můžeme omezit pomocí

$$\sum_{j=1}^{m} |A_x| \leq m 2^{m-n} < 2^m$$

pro dostatečně velké nm (abychom to lépe viděli můžeme například dosadit $m = n^c$ a obě strany zlogaritmovat, konstantu c můžeme zvolit libovolně velkou). Množina $A_x \oplus \{z_1, \ldots, z_m\}$ tedy není dostatečně velká, aby se mohla rovna $\{0,1\}^m$.

Předpokládejme na druhou stranu, že $x \in A$ a tedy $|R_x| \leq 2^{m-n}$. Pokud existují $z_1, \ldots, z_m$ tak, že

$$\text{pro každé } w \in \{0,1\}^m \text{ platí } w \oplus \{z_1, \ldots, z_m\} \not\subseteq R_x, \tag{10}$$

pak pro každé $w \in \{0,1\}^m$ existuje j tak, že $w \oplus z_j \notin R_x$ a tedy $w \notin R_x \oplus z_j$ a proto $w \in A_x \oplus z_j$; dostáváme tak, že $A_x \oplus \{z_1, \ldots, z_m\} = \{0,1\}^m$.

Zbývá ukázat, že $z_1, \ldots, z_m$ pro které platí (10) skutečně existují. Každou m-tici $z_1, \ldots, z_m$, pro kterou (10) neplatí, unikátně determinují dvojice (w, Y) takové, že $w \in \{0,1\}^m$ a $Y = w \oplus \{z_1, \ldots, z_m\} \subseteq R_x$. (Stačí si uvědomit, že v rovnici $w \oplus \{z_1, \ldots, z_m\} = S$ znalost w a S jednoznačně určuje $\{z_1, \ldots, z_m\}$.) Počet m prvkových podmnožin R_x je ovšem omezen $(2^{m-n})^m$. Máme tak, že počet (10) nevyhovujících m-tic je omezen

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

Přitom ovšem počet možností, jak zvolit $z_1, \ldots, z_m$ je právě 2^{m^2}. Musí tedy skutečně existovat volba $z_1, \ldots, z_m \in \{0,1\}^m$ pro kterou platí (10). Tímto jsme dokončili důkaz pomocného tvrzení.

Platnost předchozího tvrzení využijeme ke konstrukci Σ_2-stroje, který pro vstup x

1. pomocí existenciální alternace vybere $z_1, \ldots, z_m$.
2. pomocí univerzální alternace vygeneruje všechna $w \in \{0,1\}^m$ a pro každé ověří, jestli existuje j tak, že $w \oplus z_j \in A_x$ (tím, že pro $i = 1, \ldots, m$ spustí $M(x, w \oplus z_i)$.

Polynomická složitost plyne z toho, že $m = n^c$ a toho, že M pracuje v pol. čase. □

Důsledkem předchozí věty je že, pokud $\mathbf{P} = \mathbf{NP}$ pak $\mathbf{BPP} = \mathbf{P}$.

Není znám vztah mezi **BPP** a **NP**. Věříme, že $\mathbf{NP} \not\subseteq \mathbf{BPP}$, protože z $\mathbf{NP} \subseteq \mathbf{BPP}$ plyne $\mathbf{PH} \subseteq \mathbf{BPP}$ (důkaz tohoto vynecháme) a v důsledku předchozí věty by **PH** zkolabovala na $\mathbf{\Sigma}_2^p$.

(Přirozené) problémy v BPP

- Testování prvočíselnosti. To je v současnosti triviální, protože víme že problém patří do **P**. Nicméně, i kdyby nepatřil do **P**, patřil by do **BPP**: existují efektivní pravděpodobnostní algoritmy, např. Miller-Rabinův test, Solovay–Strassen test.
- Příkladem problému, který patří do **BPP**, ale nevíme o něm, že by patřil do **P**, je problém testování polynomických identit. Vstupem je polynom nad proměnnými $x_1, \ldots, x_n$ s celočíselnými koeficienty, který je zadán ve formě aritmetického obvodu (definice je analogická booleovskému obvodu, pouze brány místo logických operací počítají aritmetické operace $+, *, -$), a chceme zjistit, jestli je tento obvod roven 0 pro všechna ohodnocení proměnných. Klíčem k pravděpodobnostnímu algoritmu je tvrzení známé jako *Schwartz-Zippel Lemma*. Jeho důsledkem je následující tvrzení:

Věta 35

Nechť $\mathbb{F}$ je těleso a $S \subseteq \mathbb{F}$. Uvažme nenulový polynom $p(x_1, \ldots, x_n)$ s koeficienty z $\mathbb{F}$ s maximálním stupněm d. Potom pro $(s_1, \ldots, s_n)$ s uniformní pravděpodobností vybraném z S^n platí

$$\Pr[p(s_1, \ldots, s_n) = 0] \leq \frac{d}{|S|}.$$

Test tedy spočívá v navzorkování $(s_1, \ldots, s_n)$ a výpočtu hodnoty $p(s_1, \ldots, s_n)$. Pokud je nenulová, není p roven 0 pro všechna ohodnocení. Je-li hodnota rovna 0, může být p nenulový s pravděpodobností nejvýše $\frac{d}{|S|}$.

Literatura

[2](kap. 13, 14)

Reference

[1] Michael Sipser. Introduction to the Theory of Computation (2nd edition). Thomson. 2006.

[2] Dexter C. Kozen. Theory of Computation. Springer. 2006

[3] Christos H. Papadimitriou. Computational Complexity. Addison-Wesley. 1995

[4] S. Arora, B. Barak. Computational Complexity: A Modern Approach. Cambridge University Press. 2009.

[5] Stephen Cook, Phuong Nguyen. Logical Foundations of proof complexity. Cambridge University Press, 2014

Todo list