Pokročilá teória zložitosti

Kubo Kováč

Obsah

Kapitola 1

Úvod

1.1 Zložitosť problémov

Veľká časť informatiky sa zaoberá návrhom rýchlych, efektívnych a praktických riešení rôznych problémov – návrhom algoritmov a dátových štruktúr.

Pri každom novom riešení si zároveň kladieme otázku:

No dobre, ale nedalo by sa to ešte lepšie?

Intuitívne pritom chápeme, že existuje nejaká hranica a lepšie sa to už nedá... a práve tieto

„hranice možného“

skúma teória zložitosti. Ideálne by sme chceli poznať skutočnú zložitosť všetkých zaujímavých problémov, tzn. poznať najlepšie riešenia a vedieť dokázať, že lepšie už neexistujú.

Sú problémy, o ktorých vieme veľa. Napríklad medián (stredný prvok po zotriedení) dokážeme nájsť v čase $O(n)$ a očividne lepšie ako $\Omega(n)$ sa to nedá. Ba čo viac, existuje algoritmus, ktorý spraví $2.95n + o(n)$ porovnaní a vieme dokázať, že ľubovoľný algoritmus potrebuje spraviť v najhoršom prípade *aspoň* $2.01n + o(n)$ porovnaní.

Triediť (v porovnávacom modeli) dokážeme v čase $O(n \log n)$ a dá sa ukázať, že $\Omega(n \log n)$ porovnaní je potrebných. Ba čo viac, ľubovoľný porovnávací triediaci algoritmus potrebuje v najhoršom prípade aspoň $\lceil \lg n! \rceil \approx n \lg n - 1.4426n + O(\log n)$ porovnaní a existuje algoritmus, ktorý dokáže triediť na $n \lg n - 1.3999n + o(n)$ porovnaní.

Pri iných problémoch je situácia zúfalejšia: Napríklad taký 3SAT (je daná booleovská formula v konjunktívnom normálnom tvare splniteľná?) vieme riešiť triviálne v čase $O(2^n n)$ vyskúšaním všetkých možností (všetkých pravdivostných ohodnotení). Dá sa to lepšie? Áno, v súčasnosti najlepší algoritmus rieši 3SAT v čase $O(1.308^n)$ v najhoršom prípade. Dá sa to ešte lepšie? Nepoznáme dôvod, prečo by nie. Viacero ľudí však verí, že na riešenie potrebujeme exponenciálny čas. Tzn., že existuje konštanta $c > 1$ taká, že všetky algoritmy riešiace 3SAT

potrebujú aspoň $\Omega(c^n)$ času v najhoršom prípade. Toto je tzv. Hypotéza o exponenciálnom čase (ETH), dokázať ju však zatiaľ nevieme. Veľa ľudí verí, že 3SAT sa určite nedá riešiť v polynomiálnom čase – žiaľ, ani to zatiaľ nevieme vylúčiť (toto je slávny problém $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$). Situácia je ešte zúfalejšia: my nevieme vylúčiť ani len lineárny algoritmus! Najlepší známy dolný odhad je, že ak nejaký algoritmus rieši SAT v čase t a pamäti s, potom $t \times s \geq n^{1.801}$ – teda vieme vylúčiť lineárny algoritmus so sublineárnou pamäťou $O(n^{0.8})$.

Veľmi zvláštne je na tom problém faktorizácie (rozklad čísla na súčin prvočísel). Zatiaľ najlepší algoritmus má heuristicky odhadovanú zložitosť

$$\exp\left(\left(\sqrt[3]{\frac{64}{9}} + o(1)\right)(\ln N)^{\frac{1}{3}}(\ln \ln N)^{\frac{2}{3}}\right),$$

čo je (v závislosti od počtu bitov $n = \lg N$) veľmi zhruba asi $\approx 2^{O(n^{1/3}(\lg n)^{2/3})}$ – viac ako ľubovoľný polynóm ($n^{O(1)}$), ale menej ako ľubovoľná exponenciálna funkcia ($2^{\Omega(n)}$). Existuje polynomiálny algoritmus? Nikto nevie, hoci odborníci odhadujú, že skôr asi nie. Zaujímavé je, že existuje *kvantový* algoritmus, ktorý faktorizuje v čase $O(n^3)$, kde n je počet cifier N. Zatiaľ nevieme, či sa kvantové počítače dajú reálne fyzicky zostrojiť, ale je možné, že v budúcnosti sa faktorizácia bude dať prakticky riešiť.

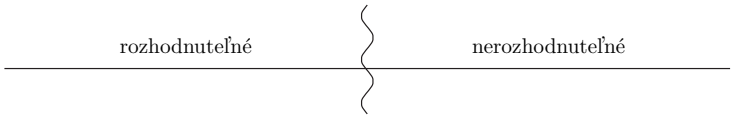
Ukazuje sa, že dokázať, že nejaký problém je (dokázateľne) ťažký je... nuž, ťažké samé o sebe... Kým na horný odhad stačí jeden algoritmus, na dolný odhad treba všetky efektívne riešenia vylúčiť.

Ukazuje sa však aj to, že relatívne úspešne dokážeme určovať „relatívnu“ zložitosť problémov – problémy vieme porovnávať a dokázať, že jeden je ťažší (alebo aspoň tak ťažký) ako iný, alebo vieme dokázať, že niektoré problémy sú zhruba rovnako ťažké. Prelomové boli v tomto články Stephena Cooka v Západnom bloku a Leonida Levina vo Východnom bloku, ktorí v roku 1971 nezávisle od seba dokázali, že SAT je jeden z tých najťažších problémov, ktoré sa dajú riešiť nedeterministicky v polynomiálnom čase – teda jeden z najťažších problémov, ktorých riešenie sa dá v polynomiálnom čase overiť. Rok na to Richard Karp ukázal, že 21 problémov, ktoré na prvý pohľad vyzerajú veľmi odlišne (napríklad SAT, problém kliky, ofarbovanie grafu, hľadanie Hamiltonovskej cesty, či problém batohu), sú v skutočnosti zhruba rovnako ťažké. Presnejšie, ak vieme jeden problém riešiť v čase $t(n)$, tak aj všetky ostatné vieme riešiť v čase $t(n^{O(1)})$, teda buď sa dajú riešiť v polynomiálnom čase všetky tieto problémy, alebo žiaden a ak na riešenie jedného problému treba exponenciálne veľa času, tak to platí pre všetky.

V teórii zložitosti sa preto snažíme hľadať súvislosti medzi rôznymi problémami, zoskupovať podobne ťažké problémy, kategorizovať ich do rôznych *tried* zložitosti a následne študujeme vzťahy medzi týmito triedami.

1.2 Triedy zložitosti

Ak sa na všetky problémy pozrieme zdiaľky, to najhrubšie delenie problémov je na rozhodnuteľné (vypočítateľné, rekurzívne) a nerozhodnuteľné.



Takto na prvú kôpku môžeme zaradiť napríklad hľadanie najkratšej cesty, na druhú problém zastavenia. Na prvú Ackermannovu funkciu, na druhú Busy Beavera. Na prvú lineárne programovanie, na druhú problém totálnosti.

Ak porozumieme tomuto základnému deleniu, môžeme sa skúsiť na problémy pozrieť bližšie a roztriediť ich do jemnejších kategórií. Teória vypočítateľnosti sa venuje najmä nerozhodnuteľným problémom a problémom „na hranici“ rozhodnuteľnosti. Aj nerozhodnuteľné problémy môžeme totiž ďalej deliť na „ľahšie“ a „ťažšie“ – niektoré problémy sú napríklad rekurzívne vyčísliteľné, niektoré ani to nie; niektoré by sa dali vypočítať, ak by sme mali čiernu krabičku (orákulum) riešiacu problém totálnosti, niektoré ani tak; niektoré problémy sa dajú aspoň popísať formulou s piatimi kvantifikátormi (a rekurzívnym predikátom), niektoré ani tak.

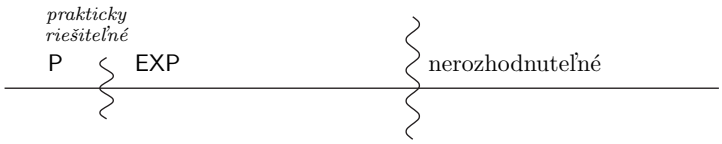
My sa naopak budeme venovať tým riešiteľným, rozhodnuteľným problémom. Budeme sa venovať výpočtom, ktoré po konečnom počte krokov skončia a bude nás zaujímať, koľko výpočtových „prostriedkov“ spotrebujú – napríklad koľko času, koľko pamäte, koľko procesorov, koľko náhodných bitov, koľko nedeterminizmu, atď.

Totiž, napriek tomu, že by sme tieto problémy teoreticky dokázali riešiť, ak by sme mali dostatok času, pamäte a energie, v praxi nie vždy dostatok času, či pamäte máme a požiadavky niektorých algoritmov môžu byť astronomické. Napríklad diagonála spomínanej Ackermannovej funkcie rastie tak rýchlo, že už $a(4) = A_{4,4} = 2^{2^{2^{65536}}} - 3$ je nepredstaviteľne veľké číslo. Meyer a Stockmeyer zase ukázali problém (zistiť, či je formula v nejakej konkrétnej teórii pravdivá), kde už na vstupy dĺžky ≈ 4000 bitov by sme dokázateľne potrebovali logické obvody s $> 10^{125}$ hradlami. Pre porovnanie, odhadovaný počet atómov v pozorovateľnom vesmíre je niekde medzi 10^{78} a 10^{82}.

V praxi nás teda najviac zaujíma, ktoré problémy sú „prakticky riešiteľné“ a ktoré nie. A je priepastný rozdiel medzi problémami, ktoré sa dajú riešiť v polynomiálnom čase a tými, ktoré sa dajú riešiť v exponenciálnom čase[1] (napríklad algoritmus, ktorý beží v čase 2^n trvá dvakrát toľko už pre n o jedna väčšie,

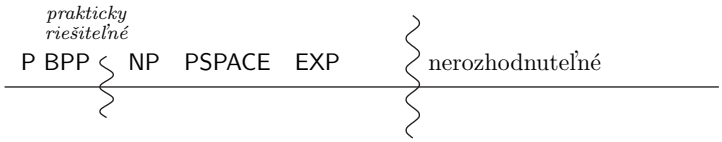
[1] Napriek očividným námietkam, že riešenie v čase $O(n^{1\,000\,000})$ nie je praktické a $O(1.000001^n)$ môže byť, je užitočné uvažovať takéto veľké a robustné triedy.

kdežto kvadratický algoritmus trvá dvakrát toľko pre 1.414-*násobne* veľké n).



Na jednej strane, medzi praktické riešenia by sme mohli zaradiť aj efektívne pravdepobnostné algoritmy (trieda BPP obsahuje problémy riešiteľné pravdepobnostne v polynomiálnom čase s iba veľmi malou chybou). Prirodzene však potom vyvstáva otázka, nakoľko nám náhodnosť pomáha a či dokážeme v polynomiálnom čase viac problémov vyriešiť pravdepobnostne ako deterministicky. Toto je otázka, či $\mathsf{P} \stackrel{?}{=} \mathsf{BPP}$, ktorej sa budeme venovať v VI. časti.

Na druhej strane, veľa zaujímavých praktických problémov sa dá riešiť skúšaním všetkých možností síce v exponenciálnom čase, ale v polynomiálnom priestore (PSPACE) a dokonca, ak by nám niekto prezradil riešenie, vedeli by sme ho v polynomiálnom čase *overiť* (trieda NP).



Už v 50-tych rokoch sa logik Kurt Gödel, áno, ten, čo dokázal, že v matematike existujú nedokázateľné tvrdania, v liste Johnovi von Neumannovi zamýšľal, aké ťažké je niečo dokázať: Ak existuje dôkaz nejakého tvrdenia dĺžky ℓ, zjavne ho vieme nájsť v exponenciálnom čase tak, že generujeme všetky reťazce dĺžky ℓ a kontrolujeme, či je to korektný dôkaz nášho tvrdenia; nedalo by sa to však v čase polynomiálnom od ℓ? Aj toto sa dá v zásade interpretovať ako zárodok otázky, či $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$.

V súčasnosti vieme dokázať, že $\mathsf{P} \neq \mathsf{EXP}$ a že v exponenciálnom čase vieme vypočítať viac ako v polynomiálnom (dôkaz v Kapitole 3). V časti IV. si dokonca ukážeme, že vymýšľanie stratégie, ako potiahnuť v niektorých stolových hrách, patrí medzi tie najťažšie problémy riešiteľné v exponenciálnom čase (a preto nemajú polynomiálne riešenie). Okrem toho si tiež ukážeme viaceré populárne hry, ktoré patria medzi tie najťažšie problémy v NP či PSPACE. Otázka, či $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$, dokonca či $\mathsf{P} \stackrel{?}{=} \mathsf{PSPACE}$ je stále otvorená, napriek tomu vieme o problémoch „niekde medzi“ P a PSPACE veľa povedať a v II. časti rozvinieme zovšeobecnenie nedeterminizmu, vďaka ktorému vieme tento terén presnejšie zmapovať.

V III. časti sa zameriame na „ľahké“ problémy riešiteľné v polynomiálnom čase a pozrieme sa na to, aké problémy sa dajú riešiť ešte efektívnejšie – veľmi rýchlo paralelne alebo s veľmi malou pamäťou (a ktoré sa naopak nedajú). V piatej časti sa zase vrátime k logike – kolíske informatiky. Ako dokázali Church a Turing, neexistuje algoritmus (ktorý vždy skončí) a rozhodne, či je nejaké

matematické tvrdenie pravdivé (tzv. *Entscheidungsproblem*). Napriek tomu, pre niektoré obmedzenejšie logiky taký algoritmus existuje. Pozor, spoiler, takýmto algoritmom to bude väčšinou trvať astronomický – hoci konečný – čas.

Na rozmyslenie

- Stretli ste sa už s nejakými dolnými odhadmi? Dajte príklad.
- Poznáte nejaké NP-úplné problémy? Čo PSPACE-úplné? Alebo EXP-úplné?

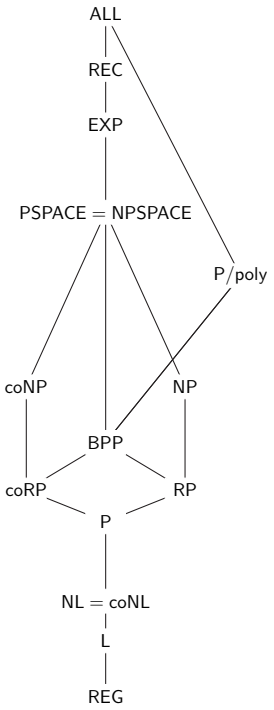
Časť I

Základy teórie zložitosti

Kapitola 2

Výpočtové modely a miery zložitosti

Táto kapitola sa dá zhrnúť v jednom obrázku:



Obr. 2.1: Základné zložitostné triedy a vzťahy medzi nimi.

2.1 Turingove stroje a RAM

V tejto knihe nás bude zaujímať časová a pamäťová zložitosť algoritmov, tzn. počet krokov výpočtu nejakého stroja a pamäť, ktorú pri tom zaberie. Na mieste sú teda otázky

- Aký model budeme používať?
- Záleží na tom?
- Ak áno, do akej miery?

Poďme pekne po poriadku: Štandardný teoretický model, ktorý budeme používať je Turingov stroj (skrátene TS) s jednou read-only páskou, na ktorej je vstup, k pracovnými páskami (kde k je ľubovoľne veľká konštanta) a prípadne jednou write-only páskou na výstup.

Definícia 2.1 (DTIME, DSPACE). *Nech $f : \mathbb{N} \to \mathbb{N}$ je funkcia. Definujeme* $\mathsf{DTIME}(f(n))$ *ako triedu jazykov rozhodnuteľných na Turingovom stroji v čase* $O(f(n))$ *a* $\mathsf{DSPACE}(f(n))$ *ako triedu jazykov rozhodnuteľných v pamäti* $O(f(n))$.

Špeciálne najznámejšie triedy, ktoré budeme študovať sú

- $\mathsf{L} = \mathsf{DSPACE}(\log n)$ – *logaritmický priestor,*
- $\mathsf{P} = \bigcup_k \mathsf{DTIME}(n^k)$ – *polynomiálny čas,*
- $\mathsf{PSPACE} = \bigcup_k \mathsf{DSPACE}(n^k)$ – *polynomiálny priestor,*
- $\mathsf{EXP} = \bigcup_k \mathsf{DTIME}(2^{n^k})$ – *exponenciálny čas.*

Zrejme výpočty v čase $t(n)$ zaberajú najviac $O(t(n))$ pamäte a na druhej strane výpočty, čo zaberajú $s(n)$ pamäte môžu bežať najviac $2^{O(s(n))}$ krokov (potom sa konfigurácie zopakujú a stroj sa zacyklí). Konkrétne z toho vyplýva, že

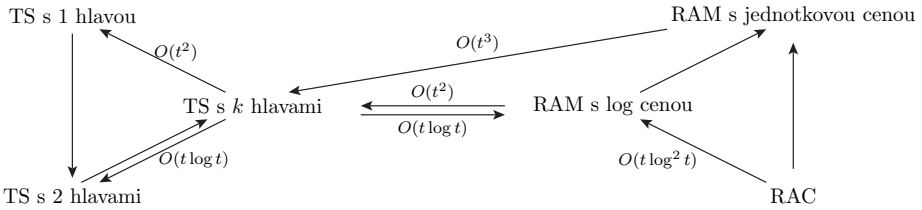
$$\mathsf{L} \subseteq \mathsf{P} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP}.$$

Sú tieto inklúzie striktné? Platí $\mathsf{L} \subset \mathsf{P} \subset \mathsf{PSPACE} \subset \mathsf{EXP}$? Dobrá otázka. Veríme, že áno, ale v súčasnosti to nevieme dokázať.

Invariancia

Záleží na modeli? Jednoznačne áno.

Napríklad ak potrebujeme sčítať pár čísiel, je rozdiel, či použijeme model ako RAM, kde sčítanie trvá konštantný čas alebo Turingov stroj, kde sčítanie musíme prácne simulovať. Ak chceme zotriediť n prvkov, je rozdiel, či to budeme robiť v porovnávacom modeli (a potrebujeme aspoň $\Omega(n \log n)$ porovnaní), alebo môžeme použiť radix sort. Ak napríklad chceme skontrolovať, či je dané slovo palindróm, je rozdiel, či použijeme Turingov stroj s dvoma hlavami alebo len s jedinou – s dvoma hlavami to ide jednoducho v lineárnom čase, jediná



Obr. 2.2: Vzájomné simulácie jednotlivých modelov. Šípka $A \xrightarrow{f(t)} B$ znamená, že výpočet A v čase t sa dá simulovať v modeli B v čase $f(t)$.

hlava musí pri porovnávaní písmen lietať hore-dole zo strany na stranu (a dá sa dokázať, že lepšie ako v kvadratickom čase sa to nedá).

Na druhej strane, záleží, aké jemné rozdiely nás zaujímajú. Napríklad v teórii vypočítateľnosti sa ukazuje, že ak chceme iba rozdeliť problémy na rozhodnuteľné a nerozhodnuteľné, na modeli až tak nezáleží: Je jedno, či použijeme Turingove stroje, λ-kalkulus, či μ-rekurzívne funkcie. Je jedno, či použijeme super-rýchly paralelný RAM, alebo superpomalý stroj s dvoma počítadlami. Je jedno, či využijeme nedeterminizmus. Všetky tieto (Turingovsky úplné) stroje definujú tie isté triedy rozhodnuteľných a nerozhodnuteľných problémov. Dôkaz: jednotlivé modely sa vedia navzájom simulovať.

A podobne to je aj pri „veľkých“ robustných triedach ako L, P, PSPACE, EXP, spomínaných vyššie. Na modeli síce záleží, ale nie až tak, pretože „rozumné“ modely sa vedia navzájom *efektívne* simulovať (pozri obr. 2.2):

Téza o invariancii 1. *Existuje štandardná trieda modelov, ktorá okrem iných obsahuje rôzne varianty Turingovych strojov, rôzne varianty RAM s logaritmickou mierou, či RAM s $O(\log n)$-bitovými registrami. Stroje z tejto triedy sa vedia navzájom simulovať, pričom výpočet v čase $t(n)$ a pamäti $s(n)$ sa dá simulovať v polynomiálnom čase $O(t(n)^k)$ s len o konštantnu horšou pamäťou $O(s(n))$.*

Triedy ako L, P, PSPACE, či EXP sú pre tieto modely rovnaké.

V RAM modeli môžeme uvažovať napr. stroj s neobmedzenými registrami $X_0, X_1, X_2, \ldots$, a inštrukciami:

inštrukcia	cena
$X_i \leftarrow c$ (kde c je konštanta)	1
$X_i \leftarrow X_j \circ X_k$, kde $\circ \in \{+, -, \wedge, \vee, \neg\}$	$\ell(X_j) + \ell(X_k)$
$X_i \leftarrow X_{X_j}$, $X_{X_j} \leftarrow X_i$	$\ell(X_j) + \ell(X_{X_j})$
if $X_i > 0$ goto m	$\ell(X_i)$
načítaj j-ty znak vstupu do X_i	ℓ(vstup)
vypíš X_i na výstup	$\ell(X_i)$
HALT/ACCEPT/REJECT	1

Časovú zložitosť definujeme ako súčet cien $\ell(n)$ jednotlivých inštrukcií.

Alternatívou k RAMu je tzv. RAC model, kde pridáme navyše aritmetické inštrukcie $*$, /, %, ale obmedzíme veľkosť registrov na $\lceil k \log n \rceil$ bitov pre ne-

jaké konštantné k (výsledok $\pm, *$ bude modulo $2^{\lceil k \log n \rceil}$). Všetky operácie majú jednotkovú cenu.

Jediné, na čo si treba dať pozor, keď definujeme model podobný RAMu, je, aby sme nemali malú (jednotkovú) cenu a zároveň príliš silné inštrukcie (napr. násobenie) a zároveň registre neobmedzenej veľkosti. Pomocou k násobení totiž vieme vytvoriť číslo 2^{2^k} (keďže $2^{2^k} \times 2^{2^k} = 2^{2^k+2^k} = 2^{2^{k+1}}$), ktoré má až exponenciálny počet bitov. Spolu s ďalšími bitovými operáciami (AND, OR, shift) by sme tak dostali príliš silný model, ktorý zvládne riešiť úlohy rýchlo vďaka bitovému paralelizmu.

2.2 Nedeterminizmus

Nedeterministický Turingov stroj (NTS) môže mať počas výpočtu v danej situácii viacero možností ako pokračovať (pre dané stavy hláv a načítané znaky je v δ-funkcii viacero možných krokov). Namiesto jedného výpočtu tak dostaneme celý strom možných výpočtov a hovoríme, že stroj akceptuje vstupné slovo, ak je aspoň jeden výpočet akceptačný.

Definícia 2.2 (NTIME, NSPACE). *Nech $f : \mathbb{N} \to \mathbb{N}$ je funkcia. Definujeme* $\mathsf{NTIME}(f(n))$ *ako triedu jazykov rozhodnuteľných na nedeterministickom Turingovom stroji v čase $O(f(n))$ a* $\mathsf{NSPACE}(f(n))$ *ako triedu jazykov rozhodnuteľných v pamäti $O(f(n))$. Presnejšie, budeme používať tzv. slabú definíciu: stačí, ak pre každé slovo, ktoré patrí do jazyka, existuje aspoň jeden výpočet v čase/pamäti $O(f(n))$.*

Špeciálne najznámejšie triedy, ktoré budeme študovať sú

- $\mathsf{NL} = \mathsf{NSPACE}(\log n)$ – *logaritmický priestor,*
- $\mathsf{NP} = \bigcup_k \mathsf{NTIME}(n^k)$ – *polynomiálny čas,*
- $\mathsf{NEXP} = \bigcup_k \mathsf{NTIME}(2^{n^k})$ – *exponenciálny čas.*

Zrejme $\mathsf{DTIME}(t(n)) \subseteq \mathsf{NTIME}(t(n)) \subseteq \mathsf{DSPACE}(t(n))$ a $\mathsf{NSPACE}(s(n)) \subseteq \mathsf{DTIME}(2^{O(s(n))})$, takže

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP} \subseteq \mathsf{EXPSPACE}.$$

Sú tieto inklúzie vlastné? Veľmi dobrá otázka. Asi áno, ale nevieme to dokázať.

Existuje však druhá možnosť:

Certifikáty

Namiesto stroja, ktorý sa nedeterministicky rozhoduje, zapíšme tieto rozhodnutia na pásku (read-only, čitateľnú iba zľava doprava). Na základe tejto pásky už vieme výpočet robiť deterministicky, takže to, že existuje nejaký nedeterministický výpočet, je ekvivalentné tvrdeniu, že existuje postupnosť rozhodnutí.

Napríklad triedu NP vieme ekvivalentne definovať:

$L \in \mathsf{NP}$, ak existuje polynomiálny *deterministický* TS M a polynóm p také, že $x \in L \iff \exists u \in \{0,1\}^{p(|x|)} : M(x,u) = 1$.

Inými slovami, problémy v NP sú tie, ktoré majú riešenia polynomiálnej dĺžky, ktoré sa dajú v polynomiálnom čase overiť.

Triedu coNP tvoria komplementy problémov v NP. Ekvivalentne, $L \in \mathsf{coNP}$, ak $x \in L \iff \forall u : M(x,u) = 1$, kde u má polynomiálnu dĺžku a M beží v polynomiálnom čase.

Operátory

Vo všeobecnosti môžeme definovať množinu certifikátov (svedkov)

$$W(p, L, x) = \{y \in \{0,1\}^{p(|x|)} \mid x\#y \in L\}.$$

a pre ľubovoľnú triedu jazykov $\mathcal{C}$ operátory $\exists$, $\forall$ a co:

$$\begin{aligned} L \in \exists \cdot \mathcal{C} &\iff \exists L' \in \mathcal{C} : x \in L \leftrightarrow |W(n^k, L', x)| > 0 \\ L \in \forall \cdot \mathcal{C} &\iff \exists L' \in \mathcal{C} : x \in L \leftrightarrow |W(n^k, L', x)| = 2^{|x|^k} \\ L \in \mathsf{co} \cdot \mathcal{C} &\iff L^C \in \mathcal{C} \end{aligned}$$

Potom

$$\begin{aligned} \mathsf{NP} &= \exists \cdot \mathsf{P} \\ \mathsf{coNP} &= \mathsf{co} \cdot \mathsf{NP} = \mathsf{co} \cdot \exists \cdot \mathsf{P} = \forall \cdot \mathsf{P} \\ \text{vo všeobecnosti } \mathsf{co} \cdot \exists \cdot \mathcal{C} &= \forall \cdot \mathcal{C} \text{ a } \mathsf{co} \cdot \forall \cdot \mathcal{C} = \exists \cdot \mathcal{C}. \end{aligned}$$

Nedeterministický priestor

Zatiaľčo o nedeterministickom čase až tak veľa nevieme (nevieme, či $\mathsf{P} \subset \mathsf{NP}$, nevieme, či $\mathsf{NP} \neq \mathsf{coNP}$), o nedeterministickom priestore vieme dve základné prekvapivé veci:

- nedeterministický priestor $s(n)$ vieme simulovať deterministicky v $s(n)^2$,
- nedeterministický priestor je uzavretý na komplement.

Veta 2.1 (Savitch (1970)). *Nech* $\log n \leq s(n)$ *je páskovo konštruovateľná, potom*

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

Takže napríklad $\mathsf{PSPACE} = \mathsf{NPSPACE} = \mathsf{coNPSPACE}$.

■ **Dôkaz.** Nech M je NTS; chceme otestovať, či sa M dostane z konfigurácie C_1 do C_2 na k krokov (zo začiatočnej do ľubovoľnej akceptačnej na $c^{s(n)}$ krokov). Ak $C_1 = C_2$ alebo $C_1 \vdash C_2$ na 1 krok, áno; v opačnom prípade, ak $k \leq 1$, tak nie; a ak $k \geq 2$, vyskúšame všetky možné prostredné konfigurácie C a rekurzívne

otestujeme, či sa M dostane z C_1 do C na $\lfloor k/2 \rfloor$ a z C do C_2 na $\lceil k/2 \rceil$ krokov. □

S nasledujúcou vetou sa čitateľ možno stretol na Formálnych jazykoch a automatoch v tvare, že kontextové jazyky sú uzavreté na komplement. Kontextové jazyky sú však práve jazyky akceptované nedeterministickým lineárne ohraničeným automatom, čo je trieda $\mathsf{NSPACE}(n)$. Tento dôkaz sa dá zovšeobecniť pre $s(n) \geq \log n$:

Veta 2.2 (Immerman (1988), Szelepcsényi (1988)). *Nedeterministický priestor je uzavretý na komplement:* $\mathsf{NSPACE}(s(n)) = \mathsf{coNSPACE}(s(n))$ *pre* $s(n) \geq \log n$. *Špeciálne* $\mathsf{NL} = \mathsf{coNL}$ *a kontextové jazyky sú uzavreté na komplement.*

2.3 Booleovské obvody

Booleovské obvody sú veľmi prirodzený matematický model pre digitálne elektronické obvody, z ktorých sú poskladané mikročipy v našich počítačoch. Príklady takýchto obvodov vidíme na obrázku 2.3.

Definícia 2.3 (BO). *Booleovský obvod C je acyklický orientovaný graf; vrcholy, do ktorých nejde hrana voláme vstupné, ostatné vrcholy sú hradlá označené $\wedge$, $\vee$, alebo $\neg$, niektoré vrcholy sú označené ako výstupné. Tradične budeme vyžadovať, že hradlá $\wedge$ a $\vee$ majú dva vstupy, hradlo $\neg$ jeden. V booleovských obvodoch s neobmedzeným stupňom môžu mať hradlá $\wedge$ a $\vee$ ľubovoľne veľa vstupov. Hodnota obvodu $C(x)$ sa vypočíta prirodzeným spôsobom – vstupným hradlám sa priradia hodnoty, postupne sa aplikujú jednotlivé hradlá a vypočítajú výstupné hodnoty.*

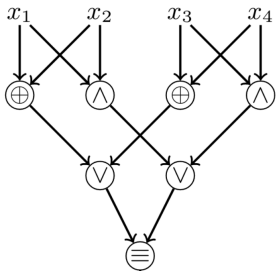
Tak ako pre Turingove stroje nás zaujímali čas a priestor, pre booleovské obvody nás budú zaujímať najmä dve miery zložitosti:

- veľkosť – počet hradiel obvodu, značíme $|C|$ a
- hĺbka – dĺžka najdlhšej cesty medzi vstupným a výstupným hradlom.

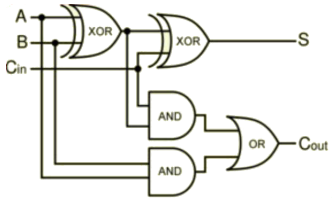
Veľkosť súvisí s časom, za ktorý by sme obvod vyhodnocovali sekvenčne, po jednotlivých hradlách, respektíve s celkovou prácou, ak by sme obvod vyhodnocovali paralelne. Hĺbka súvisí s časom, za ktorý môžeme obvod vyhodnotiť paralelne.

Všimnite si, že jeden konkrétny obvod má fixný počet vstupov. Takto napríklad môžeme študovať problém sčítania 32-bitových čísel. Ak však chceme model, ktorý funguje pre ľubovoľnú veľkosť vstupu, budeme uvažovať postupnosť $C_0, C_1, C_2, \ldots$ obvodov, pričom C_n je obvod, ktorý funguje na n-bitových vstupoch.

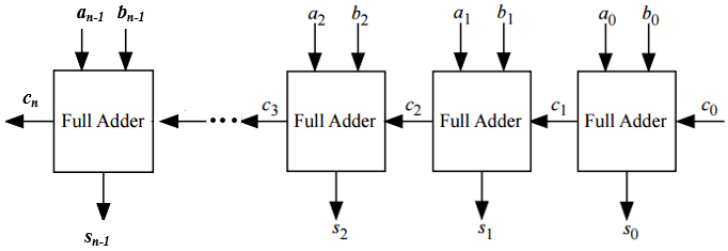
Navyše navrhované obvody $C_0, C_1, C_2, \ldots$ väčšinou nie sú úplne náhodné, „hala-bala“, ale nasledujú podľa nejakého vzoru, schémy. Napríklad na obr. 2.3c je zobrazená schéma obvodu, ktorý sčíta dve n-bitové čísla. Vo všeobecnosti hovoríme, že postupnosť obvodov je *uniformná*, ak pre dané n vieme efektívne zostrojiť obvod C_n. Konkrétne



(a) Najmenší obvod (ak povolíme hradlo XOR ($\oplus$) a ekvivalenciu ($\equiv$)), ktorý testuje, či je počet bitov deliteľný tromi, teda či $x_1 + x_2 + x_3 + x_4 \equiv 0 \pmod 3$.



(b) Úplná 1-bitová sčítačka; hodnota S je súčet $A + B + C_{in} \pmod 2 = A \oplus B \oplus C_{in}$, $C_{out} \equiv (A + B + C_{in} \geq 2) \equiv C_{in} \wedge (A \oplus B) \vee (A \wedge B)$ je prenos do vyššieho rádu.



(c) Sčítanie n-bitových čísel sa dá potom poskladať z 1-bitových sčítačiek.

Obr. 2.3: Príklady booleovských obvodov.

Definícia 2.4 (Logspace uniformnosť). *Hovoríme, že postupnosť obvodov $\{C_n\}_{n\in\mathbb{N}}$ je logspace uniformná, ak pre dané n (resp. slovo 1^n) vieme vypočítať popis C_n s použitím logaritmickej pamäte. (Z toho mimochodom vyplýva, že obvod C_n je polynomiálne veľký od n. Pre väčšie obvody treba uvažovať iné definície uniformnosti.)*

Veta 2.3. *Trieda jazykov rozhodovaná logspace uniformnými obvodmi je práve trieda* P.

■ **Dôkaz.** Prenechávame čitateľovi. □

2.4 Paralelizmus

Ako sme naznačili vyššie, dobrý model pre štúdium paralelizmu sú obvody. Praktickejší na programovanie je však model PRAM (paralelný RAM), ktorý sa skladá z $p(n)$ procesorov (definovaných ako RAM s jednotkovou cenou vyššie) a zo zdieľanej pamäte – registrov $G_0, G_1, \ldots$. Navyše každý procesor má svoje lokálne registre $X_{i,0}, X_{i,1}, \ldots$; i-ty procesor má svoj identifikátor $i \in \{1, \ldots, p(n)\}$ uložený v špeciálnom registri *pid* a každý procesor pozná hodnotu $p(n)$.

Procesory sú synchronizované, začnú naraz a v každom kroku vykonajú buď lokálnu operáciu (ako v modeli RAM), alebo zápis alebo čítanie z globálnej pamäte. Podľa prístupu ku globálnej pamäti potom rozlišujeme varianty:

- CRCW (concurrent read, concurrent write) – viacero procesorov môže čítať z aj zapisovať do toho istého registra; načíta sa hodnota pred zmenou, zapíše sa jedna z hodnôt (v modeli PRIORITY CRCW sa zapíše hodnota procesoru s najmenším id, v modeli ARBITRARY CRCW sa vyberie niektorá a programátor nemá kontrolu nad tým, ktorá a v modeli COMMON CRCW musia byť všetky hodnoty zapisované v jednom kroku do jedného registra rovnaké, v opačnom prípade je výpočet nedefinovaný).

- CREW (concurrent read, exclusive write) – viacero procesorov môže čítať, zapisovať môže len jeden

- EREW (exclusive read, exclusive write) – žiadne dva procesory nemôžu čítať alebo zapisovať do toho istého registra.

Vyššie sme spomínali, že ak sekvenčnému RAMu pridáme násobenie, povolíme neobmedzené registre a napriek tomu budeme uvažovať jednotkovú cenu inštrukcií, dostaneme silný paralelný model.

Podobne ako pri sekvenčných, aj pri paralelných výpočtoch teda vyvstáva otázka, aký model zvoliť a do akej miery na tom záleží. Aj v tomto prípade existuje veľká trieda modelov, ktoré sa dokážu navzájom efektívne simulovať (čas je polynomiálny od času simulovaného výpočtu, často dokonca nanajvýš logaritmický faktor navyše). Konkrétne (pozri obr. 2.4) napríklad:

- Hradlo s n vstupmi sa dá simulovať obvodom s binárnymi hradlami v tvare binárneho stromu hĺbky $\log n$ a veľkosti $n - 1$. Obvody s neobmedzeným stupňom veľkosti $s(n)$ a hĺbky $d(n)$ sa takto dajú simulovať obvodmi s obmedzeným stupňom hĺbky $d(n) \log s(n)$ (v prípade polynomiálnej veľkosti to je faktor $O(\log n)$ navyše) a veľkosti $O(s(n)^2)$.

- Pre CRCW PRAM s $p(n)$ procesormi bežiaci v čase $t(n)$ existuje ekvivalentná postupnosť obvodov *s neobmedzeným stupňom* hĺbky $O(t(n))$ a veľkosti $O(\mathsf{poly}(p(n), t(n), n))$ (Stockmeyer a Vishkin, 1984).

- Jeden krok CRCW PRAM s p procesormi sa dá simulovať na EREW PRAM v $O(\log p)$ krokoch. Teda výpočet CRCW PRAM v čase $t(n)$ na $p(n)$ procesoroch sa dá simulovať v čase $O(t(n) \log p(n))$ na EREW PRAM

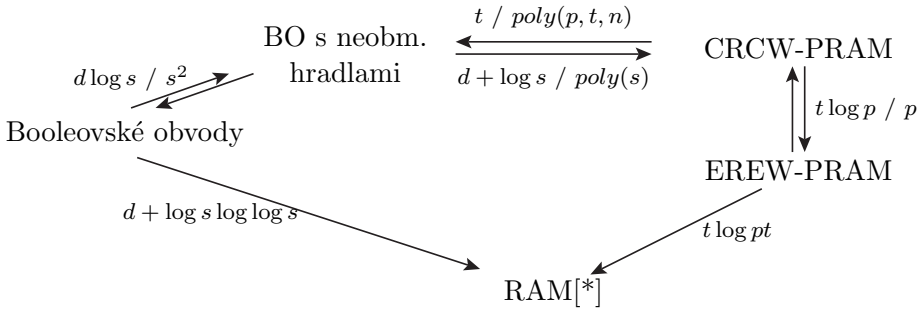
s $p(n)$ procesormi (v prípade polynomiálneho počtu procesorov to je faktor $\log n$ navyše).

- Logspace uniformné obvody hĺbky $d(n)$ a veľkosti $s(n)$ vieme simulovať na CRCW PRAM v čase $O(d(n) + \log s(n))$ a $\mathsf{poly}(s(n))$ procesormi (z toho $\log s(n)$ trvá vygenerovať a dekódovať booleovský obvod a $d(n)$ samotná simulácia) (Stockmeyer a Vishkin, 1984).

- „Dostatočne uniformné“ obvody s obmedzeným stupňom hĺbky $d(n)$ a veľkosti $s(n)$ sa dajú akceptovať na RAM[$*$] resp. RAM[$<<$, $>>$] (t.j. sekvenčný RAM s jednotkovou cenou, kde pridáme násobenie, resp. bitový posun vpravo a vľavo) v čase $O(d(n) + \log s(n) \log\log s(n))$. (Trahan a spol., 1992).

- Pre $t(n) \geq \log n$ jazyk akceptovaný na PRAM s $p(n)$ procesormi v čase $t(n)$ sa dá akceptovať na RAM[$*$] v čase $O(t(n) \log p(n) t(n))$ (Trahan a spol., 1992).

- RAM[$*$], dokonca PRAM[$*$] sú veľmi silné modely – vieme ich simulovať na PRAM v čase $t^2/\log t(n)$, ale len s použitím exponenciálneho počtu procesorov.



Obr. 2.4: Vzájomné simulácie paralelných modelov. Šípka $A \xrightarrow{f\ /\ g} B$ znamená, že výpočet A sa dá simulovať v modeli B, pričom f je hĺbka obvodu alebo paralelný čas a g je veľkosť obvodu alebo počet procesorov; pre obvody je d hĺbka a s veľkosť obvodu, pre PRAM je t čas a p počet procesorov.

Téza o paralelných výpočtoch 1. *Čas na „rozumnom“ paralelnom modeli a priestor na „rozumnom“ sekvenčnom modeli sú zhruba ekvivalentné (až na polynomiálny faktor). Špeciálne problémy riešiteľné paralelne v polynomiálnom čase sú práve tie riešiteľné sekvenčne v polynomiálnom priestore.*

V druhej časti si ukážeme ďalší model pre paralelné výpočty: alternujúce Turingove stroje.

2.5 Neuniformnosť

Ak z definície booleovských obvodov vypustíme podmienku, že n-tý obvod vieme efektívne zostrojiť, ostane nám jednoduchší, zato oveľa silnejší neuniformný model.

Definícia 2.5 (SIZE). *Definujeme* $\mathsf{SIZE}(s(n))$ *ako triedu všetkých jazykov* L*, pre ktoré existuje postupnosť obvodov* $\{C_n\}$ *veľkosti* $s(n)$ *taká, že pre každé* $x \in \{0,1\}^*$ *je* $x \in L \iff C_{|x|}(x) = 1$.

Takáto neuniformnosť je veľmi silná vec – všimnite si, že $\mathsf{SIZE}(O(n2^n))$ obsahuje *všetky* – aj nerozhodnuteľné – jazyky. Stačí totiž pre každé n do obvodu „zadrátovať“ pravdivostnú tabuľku funkcie $[x \in L]$:

$$C_n(x) \equiv \bigvee_{|z|=n, z\in L} \left(\bigwedge_{i:z_i=1} x_i \wedge \bigwedge_{i:z_i=0} \neg x_i \right) \equiv \bigwedge_{|z|=n, z\notin L} \left(\bigvee_{i:z_i=1} \neg x_i \vee \bigvee_{i:z_i=0} x_i \right)$$

Ak sa obmedzíme na polynomiálne veľké obvody, dostaneme tzv. triedu $\mathsf{P/poly}$:

Definícia 2.6 (P/poly). $\mathsf{P/poly} = \bigcup_k \mathsf{SIZE}(n^k)$ *je trieda jazykov rozhodovaných neuniformnými obvodmi polynomiálnej veľkosti.*

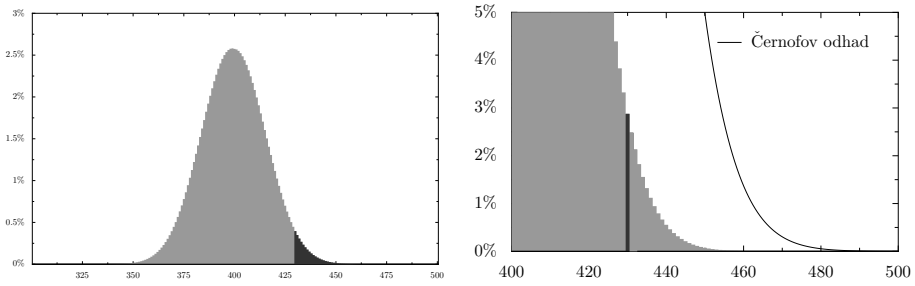
Zjavne $\mathsf{P} \subseteq \mathsf{P/poly}$. Navyše táto inklúzia je ostrá, keďže $\mathsf{P/poly}$ obsahuje nerozhodnuteľné jazyky. Stačí totiž zobrať váš obľúbený nerozhodnuteľný jazyk (napríklad problém totálnosti) a zakódovať ho unárne, nad jednopísmenovou abecedou (definujeme $L \subseteq \{1\}^*$, $1^n \in L$ práve vtedy, keď n-tý Turingov stroj zastaví na každom vstupe). Takéto jazyky sú tiež nerozhodnuteľné, avšak *všetky unárne* jazyky patria do $\mathsf{P/poly}$ – odpoveď pre každé n jednoducho zadrátujeme do obvodu. Výsledkom bude postupnosť triviálnych obvodov $C_0, C_1, C_2, \ldots$, kde C_n jednoducho povie Áno, ak $1^n \in L$ a Nie, ak $1^n \notin L$. Táto postupnosť sa, samozrejme, pre nerozhodnuteľné jazyky nedá efektívne zostrojiť, ale existovať existuje.

Turingove stroje s radou

Iný spôsob, ako môžeme definovať neuniformné modely: Predstavme si, že Turingov stroj dostane okrem vstupu dĺžky n aj radu α_n (táto rada je rovnaká pre všetky vstupy dĺžky n). Model je teda špecifikovaný jednak programom M a jednak postupnosťou rád $\alpha_0, \alpha_1, \alpha_2, \ldots$ a stroj akceptuje vstupné slovo x, ak $M(x, \alpha_{|x|}) = 1$.

Označme $\mathsf{DTIME}(t(n))/a(n)$ triedu jazykov rozhodnuteľných v čase $t(n)$ s radou dĺžky $a(n)$. Potom platí:

Veta 2.4. $\mathsf{P/poly} = \bigcup_{k,d} \mathsf{DTIME}(n^k)/n^d$*, teda neuniformné polynomiálne obvody sú ekvivalentné polynomiálnym Turingovym strojom s polynomiálnou radou.*



(a) Graf $\Pr[H = k]$. Pravdepodobnosť, že padne k hláv má binomiálnu distribúciu.

(b) Graf $\Pr[H \geq k]$. Pravdepodobnosť, že padne $\geq k$ hláv klesá exponenciálne rýchlo.

Obr. 2.5: Mincu, kde hlava padá s pravdepodobnosťou 40% hodíme 1000-krát. V priemere padne približne 400 hláv. Aká je pravdepodobnosť, že padne aspoň 500 hláv? Veľmi malá. Už pre 430 hláv je pravdepodobnosť menej ako 2.9%.

■ **Dôkaz.** Radu vieme „zadrátovať“ do neuniformného obvodu a naopak, Turingov stroj môže simulovať obvod, ktorý mu posunieme ako radu. □

2.6 Náhodnosť

Definícia 2.7. *Pravdepodobnostný Turingov stroj (PTS) definujeme podobne ako nedeterministický TS, s tým rozdielom, že PTS má v každom kroku na výber najviac dva prechody ($|\delta(q,a)| \leq 2$) a v prípade dvoch možností si hodí mincou a vyberie náhodne. To znamená, že ten istý PTS M na tom istom vstupe x môže niekedy akceptovať a niekedy odmietať. Má zmysel baviť sa o pravdepodobnosti, že M akceptuje x.*

Definícia 2.8 (BPP). BPP *je trieda jazykov L, pre ktoré existuje PTS M bežiaci v polynomiálnom čase, ktorý slová v jazyku L akceptuje s pravdepodobnosťou aspoň $2/3$ a slová mimo jazyka akceptuje s pravdepodobnosťou najviac $1/3$ (a odmieta s pravdepodobnosťou aspoň $2/3$). Hovoríme, že M akceptuje L s chybou $\varepsilon < 1/2$, ak $\Pr_r[M(x) \neq L(x)] \leq \varepsilon$ (kde jazyk L chápeme ako funkciu $z\ \Sigma^* \to \{0,1\}$). BPP-jazyky sú tie, ktoré vieme akceptovať polynomiálnym PTS s chybou najviac $1/3$.*

Voľba konštanty $1/3$ je viac-menej arbitrárna a čokoľvek menej ako $1/2$ by bolo rovnako dobré. Dokonca by stačila pravdepodobnosť chyby aspoň nezanedbateľne menšia ako polovica: $1/2 - 1/\mathsf{poly}(n)$ – opakovaním totiž vieme chybovosť exponenciálne znížiť:

Veta 2.5. *Nech L je jazyk a M polynomiálny Turingov stroj taký, že $\Pr_r[M(x,r) \neq L(x)] \leq 1/2 - 1/n^c$. Potom $L \in \mathsf{BPP}$ a existuje polynomiálny TS M' taký, že $\Pr_r[M(x,r) \neq L(x)] \leq 1/2^{n^k}$.*

■ **Dôkaz.** Stačí výpočet viackrát zopakovať a vybrať si tú častejšiu odpoveď. Označme $p = 1/2 - 1/n^c$ pravdepodobnosť chyby a $q = 1 - p$. Aká je pravdepodobnosť, že sa pomýlime v aspoň polovici prípadov? To je rovnaká úloha, ako keď máme falošnú mincu, kde hlava padá s pravdepodobnosťou p, hodíme ju t-krát a chceme vedieť, s akou pravdepodobnosťou padne aspoň polovica hláv (pozri obrázok 2.5).

Pravdepodobnosť, že sa pri $2t$ opakovaniach pomýlime práve i-krát (hodíme práve i hláv) má binomiálnu distribúciu $\binom{2t}{i}p^i q^{2t-i}$, pričom $p^i q^{2t-i} \leq p^t q^t$. Pravedepodobnosť, že sa pomýlime aspoň t-krát je najviac

$$\sum_{i=t}^{2t} \binom{2t}{i} (pq)^t \leq 2^{2t} \times (pq)^t = (4pq)^t = \left(1 - \frac{1}{n^{2c}/4}\right)^t.$$

Pre $t = n^{2c}/4$ dostávame pravdepodobnosť $(1 - 1/t)^t \leq 1/e$. Pre t ešte n^k-krát väčšie bude pravdepodobnosť omylu exponenciálne malá: $(1-1/t)^{t \times n^k} \leq 1/e^{n^k}$.

Alternatívny dôkaz je s použitím Černofovej nerovnosti (pozri sekciu A.3): Pre $t = n^{2c+k}$ je očakávaný počet hláv $n^{2c+k}/2 - n^{c+k}$. Aká je pravdepodobnosť, že to bude aspoň $(1 + \delta)$-násobok pre $\delta = 2/n^c$?

$$\Pr[X \geq (1+\delta)\mu] < e^{-\mu\delta^2/3} = e^{-\Theta(n^{2c+k})/\Theta(n^c)^2} = e^{-\Theta(n^k)}. \qquad \square$$

Špeciálne prípady pravdepodobnostných algoritmov sú tie, ktoré sa mýlia iba na jednu stranu (RP a coRP) a tie, ktoré sa nemýlia vôbec (ZPP).

Definícia 2.9 (RP, coRP, ZPP). RP *je trieda jazykov L, pre ktoré existuje PTS M bežiaci v polynomiálnom čase (PPTS), ktorý akceptuje slová v jazyku L s pravdepodobnosťou aspoň $1/2$ a slová mimo jazyka vždy odmietne. (To znamená, že ak M povie* Áno*, odpoveď je správna. Ak povie* Nie*, môže sa mýliť.)*

Naopak, pri coRP *existuje PPTS M, ktorý slová z L vždy akceptuje a slová mimo odmietne s pravdepodobnosťou aspoň $1/2$.*

ZPP *je trieda jazykov L, pre ktoré existuje PPTS M, ktorý odpovie vždy správne, alebo povie* Neviem*, pričom pravdepodobnosť, že odpovie* Neviem *je najviac $1/2$. (*ZPP *algoritmy sa prezývajú tiež Las Vegas algoritmy.)*

Zopár triviálnych inklúzií a rovností, ktoré prenechávame ako cvičenie čitateľovi:

Veta 2.6. *Platí:*

- $\mathsf{P} \subseteq \mathsf{ZPP} \subseteq \mathsf{RP} \cup \mathsf{coRP} \subseteq \mathsf{BPP} \subseteq \mathsf{PSPACE}$,
- $\mathsf{coRP} = \mathsf{co} \cdot \mathsf{RP}$, *teda v* coRP *sú doplnky jazykov v* RP,
- $\mathsf{RP} \subseteq \mathsf{NP}$, $\mathsf{coRP} \subseteq \mathsf{coNP}$,
- $\mathsf{ZPP} = \mathsf{RP} \cap \mathsf{coRP}$,
- ZPP *je trieda jazykov, pre ktoré existuje PTS M, ktorý dá vždy správnu odpoveď a pre každý vstup je očakávaná časová zložitosť polynomiálna. (To znamená, že s malou pravdepodobnosťou môže výpočet trvať aj oveľa dlhšie, ale v priemere cez všetky hody mincou je polynomiálny.)*

Certifikáty

Tak ako sme nedeterministické rozhodnutia zapísali na špeciálnu pásku a triedu sme NP definovali cez overovanie certifikátov, môžeme aj náhodné bity napísať na špeciálnu pásku a definovať BPP cez certifikáty:

Definícia 2.10 (BPP). BPP *je trieda jazykov L, pre ktoré existuje deterministický TS M pracujúci v polynomiálnom čase taký, že pre náhodné r (polynomiálnej veľkosti),*

- *ak $x \in L$, tak $\Pr_r[M(x,r) = 1] \geq 2/3$ a*
- *ak $x \notin L$, tak $\Pr_r[M(x,r) = 1] \leq 1/3$.*

Definícia 2.11 (RP, coRP). RP *je trieda jazykov L, pre ktoré existuje deterministický TS M pracujúci v polynomiálnom čase taký, že pre náhodné r (polynomiálnej veľkosti),*

- *ak $x \in L$, tak $\Pr_r[M(x,r) = 1] \geq 1/2$ a*
- *ak $x \notin L$, tak $\Pr_r[M(x,r) = 1] = 0$.*

Operátory

Vo všeobecnosti môžeme definovať operátory R a BP:

$$\begin{aligned} L \in \mathsf{R} \cdot \mathcal{C} \iff & \exists L' \in \mathcal{C} : x \in L \to |W(n^k, L', x)| \geq \tfrac{1}{2} \cdot 2^{|x|^k} \\ & \wedge\, x \notin L \to |W(n^k, L', x)| = 0 \\ L \in \mathsf{BP} \cdot \mathcal{C} \iff & \exists L' \in \mathcal{C} : x \in L \to |W(n^k, L', x)| \geq \tfrac{2}{3} \cdot 2^{|x|^k} \\ & \wedge\, x \notin L \to |W(n^k, L', x)| \leq \tfrac{1}{3} \cdot 2^{|x|^k} \end{aligned}$$

Platí: $\mathsf{RP} = \mathsf{R} \cdot \mathsf{P}$, $\mathsf{BPP} = \mathsf{BP} \cdot \mathsf{P}$.

Dobrá rada je viac ako náhoda

Predpokladá sa, že $\mathsf{BPP} = \mathsf{P}$, tzn. každý pravdepodobnostný TS vieme „derandomizovať“, pričom zložitosť sa nezhorší viac ako polynomiálne. Zatiaľ je však otázka $\mathsf{P} = \mathsf{BPP}$ otvorený problém (pozri časť VI).

To, čo vieme dokázať bezpodmienečne je, že neuniformnosť je silnejšia ako náhodné bity:

Veta 2.7 (Adleman (1978)). $\mathsf{BPP} \subseteq \mathsf{P/poly}$.

■ **Dôkaz.** Majme BPP stroj M, ktorý sa na n-bitovom vstupe pomýli s pravdepodobnosťou najviac $2^{-(n+1)}$. Pravdepodobnosť, že sa M pomýli na *aspoň jednom* n-bitovom vstupe (pre náhodný reťazec) je najviac $2^n \cdot 2^{-(n+1)} = 1/2$.

Teda aspoň polovica náhodných reťazcov je taká, že M sa nepomýli *na žiadnom vstupe* – ľubovoľný z nich môžeme zobrať ako radu pre M. □

Táto veta má aj praktické dôsledky. Vezmime napríklad také testovanie prvočíselnosti. Hoci v súčasnosti už vieme rozoznávať prvočísla deterministicky v polynomiálnom čase, omnoho rýchlejšie sú pravdepodobnostné algoritmy. Tie však s malou radou (pre danú veľkosť vstupu) dokážeme derandomizovať.

Miller-Rabinov test prvočíselnosti (Rabin, 1980) funguje nasledovne:

1. Zapíšeme $n-1$ v tvare $2^s d$, kde d je nepárne. Zvolíme náhodné $0 < a < n$, tzv. *bázu*.

2. Ak $a^d \equiv 1 \pmod{n}$, alebo pre nejaké $0 \leq k < s$ je $a^{2^k d} = a^{(n-1)/2^{s-k}} \equiv -1 \pmod{n}$, n je asi prvočíslo. V opačnom prípade je n zaručene zložené.

Test je založený na dvoch vlastnostiach prvočisel:

1. Malá Fermatova veta: $a^{p-1} \equiv 1 \pmod{p}$,

2. $\mathbb{Z}_p$ je pole, takže $x^2 \equiv 1 \pmod{p}$ má 2 riešenia: ± 1.

Ak najskôr umocníme a^d a potom s-krát na druhú, na koniec by sme mali dostať $a^{n-1} \equiv 1$. Jednotku však môžeme dostať iba tak, že už od začiatku (a^d) boli samé jednotky, alebo pred prvou jednotkou bola -1.

Dá sa dokázať, že ak je n zložené, týmto testom prejde len pre štvrtinu a-čok. Ak bázu vyberieme náhodne, prevdepodobnosť omylu je $\leq 1/4$; ak test zopakujeme k-krát s rôznymi náhodnými a-čkami, pravdepodobnosť sa zmenší na $\leq 1/4^k$.

Ak by sme testovali len m-bitové čísla, tak podľa Adlemanovej vety existuje konkrétnych $m/2$ báz, ktoré stačí vyskúšať. To je však len horný odhad.

- Pre $n < 1\,050\,535\,501$ (menej ako zhruba miliardu) stačí vyskúšať dve bázy: 336 781 006 125 a 9 639 812 373 923 155 (Izykowski a Panasiuk)

- Ak chceme otestovať 32-bitové čísla, stačí overiť bázy $a \in \{2, 7, 61\}$ (Jaeschke) a

- pre 64-bitové čísla stačí overiť bázy $\{2, 325, 9375, 28178, 450775, 9780504, 1795265022\}$ (Sinclair).

- ak za a vyskúšame prvých 13 prvočísel, dostaneme správnu odpoveď pre $n < 3\,317\,044\,064\,679\,887\,385\,961\,981$ (zhruba 24-ciferné čísla; Sorenson a Webster).

Úlohy

- Ako by ste definovali nedeterministický RAM model?

- Dokážte, že *nedeterministický* 2-páskový TS dokáže simulovať k-páskový NTS len s lineárnym spomalením, teda pre každý k-páskový NTS M bežiaci v čase $t(n)$ existuje ekvivalentný 2-páskový NTS bežiaci v čase $O(t(n))$.

- Silne nedeterministický TS je NTS, ktorý dáva tri možné odpovede: Áno/ Nie/Neviem. Hovoríme, že takýto stroj rozoznáva jazyk L, ak pre každé $x \in L$ žiadny výpočet nepovie NIE a existuje výpočet, ktorý povie Áno a naopak pre všetky $x \notin L$ žiadny výpočet nepovie Áno, ale existuje výpočet, ktorý povie Nie. Dokážte, že trieda jazykov, ktorú rozoznáva silne NTS v polynomiálnom čase je $\mathsf{NP} \cap \mathsf{coNP}$.

- Dokážte, že logspace uniformné obvody rozhodujú práve triedu P.

- Pri triede BPP je dôležité, že algoritmus dá správnu odpoveď s pravdepodobnosťou nezanedbateľne väčšou ako 1/2. Definujme triedu PP jazykov L, pre ktoré existuje polynomiálny PTS M taký, že

  - ak $x \in L$, tak $\Pr_r[M(x,r) = 1] > 1/2$ a
  - ak $x \notin L$, tak $\Pr_r[M(x,r) = 1] \leq 1/2$.

  Dokážte, že táto trieda je oveľa oveľa silnejšia: obsahuje celé NP aj coNP.

- (Parberry, 1986) Model CRCW-PRAM s neobmedzene veľkými registrami a operáciami $+, -, \times, /, \mathrm{mod}$ v konštantnom čase nie je realistický. V takomto modeli sa dá vyriešiť *akýkoľvek rekurzívny* problém v konštantnom čase(!), ak máme dosť veľa procesorov a veľké registre. Skúste to dokázať. (Hint: Ak je problém riešiteľný v čase $t(n)$, budeme potrebovať až $2^{O(t(n)^2}$ procesorov a $t(n)^2$-bitové registre. Predpokladajme, že n-bitový vstup je uložený po jednotlivých bitoch v zdieľaných registroch $G_1, \ldots, G_n$, v G_0 je dĺžka vstupu n. Ako môže takýto PRAM vôbec načítať celý vstup v konštantnom čase?).

Literatúra

Adleman, Leonard. 1978. "Two theorems on random polynomial time." In *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*. IEEE, s. 75–83.

Immerman, Neil. 1988. "Nondeterministic space is closed under complementation." *SIAM Journal on computing* 17(5), s. 935–938.

Parberry, Ian. 1986. "Parallel speedup of sequential machines: A defense of parallel computation thesis." *ACM SIGACT News* 18(1), s. 54–67.

Rabin, Michael O. 1980. "Probabilistic algorithm for testing primality." *Journal of number theory* 12(1), s. 128–138.

Savitch, Walter J. 1970. "Relationships between nondeterministic and deterministic tape complexities." *Journal of computer and system sciences* 4(2), s. 177–192.

Stockmeyer, Larry a Uzi Vishkin. 1984. "Simulation of parallel random access machines by circuits." *SIAM Journal on Computing* 13(2), s. 409–422.

Szelepcsényi, Róbert. 1988. "The method of forced enumeration for nondeterministic automata." *Acta Informatica* 26(3), s. 279–284.

Trahan, Jerry L, Michael C Loui, a Vijaya Ramachandran. 1992. "Multiplication, division, and shift instructions in parallel random access machines." *Theoretical computer science* 100(1), s. 1–44.

Kapitola 3

Hierarchie

Je pravda, že

- ak máme viac času, vieme riešiť viac problémov?
- a ak máme viac pamäte, vieme riešiť viac problémov?

Odpoveď ÁNO je intuitívne tak jasná a zrejmá, až človeka šokuje, že správna odpoveď je NIE.

Dôvodov je viac:

1. Turingove stroje s pamäťou $o(\log\log n)$ (napríklad $O(\log\log\log n)$) akceptujú iba regulárne jazyky, teda pamäť asymptoticky menšia ako $\log\log n$ je ekvivalentná konštantnej pamäti.
2. Ľubovoľný Turingov stroj s časom $\omega(n)$ vieme lineárne zrýchliť. Napríklad na n^2 krokov vieme vyriešiť presne tie isté problémy ako na $100n^2$ krokov – $100\times$viac času nám nijak nepomôže.
3. V časovej (aj pamäťovej) zložitosti dokonca existujú ľubovoľne veľké diery: Existuje napríklad časová zložitosť $t(n)$ taká, že všetky problémy riešiteľné v čase $2^{t(n)}$ sú riešiteľné aj v čase $t(n)$.

Poďme si tieto tvrdenia dokázať.

3.1 Medzery v zložitosti

Veta 3.1. $\mathsf{DSPACE}(o(\log\log n)) = \mathsf{DSPACE}(1) = \mathsf{REG}$.

■ **Dôkaz.** Bez újmy na všeobecnosti uvažujme Turingov stroj M s jednou pracovnou páskou s pamäťou $s(n)$. Počet všetkých konfigurácií, kde uvažujeme pracovnú pásku, stav a pozíciu pracovnej hlavy (ale nie pozíciu čítacej vstupnej hlavy), je $N = q \cdot s(n) \cdot d^{s(n)} \leq k^{s(n)}$, kde k je konštanta.

Uvažujme celý výpočet stroja na vstupe x a zapíšme si zoznam konfigurácií, keď stroj prechádza medzi i-tym a $(i+1)$-vým políčkom vstupu (jedným alebo druhým smerom). Takúto postupnosť voláme prechodová postupnosť p_i.

Predpokladáme, že M vždy zastane a preto žiadna prechodová postupnosť nemôže obsahovať dve rovnaké konfigurácie v tom istom smere (opačný prípad znamená, že stroj sa zacyklil). To znamená, že prechodová postupnosť má dĺžku najviac $2N$ a počet všetkých možných prechodových postupností je najviac $(2N)^{2N} \leq 2^{2^{c \cdot s(n)}}$ pre vhodnú konštantu c.

Uvedomme si, že ak máme dve miesta s rovnakými prechodovými postupnosťami, môžeme časť vstupu medzi nimi vynechať a stroj si to vôbec nevšimne (kratší vstup akceptuje práve vtedy, keď akceptuje ten dlhší).

Predpokladajme, že M používa viac ako len konštantne veľa pamäte. Tzn. pre každé k existuje vstup, na ktorom výpočet použije aspoň k políčok pásky. Existuje teda nekonečná postupnosť vstupov $x_1, x_2, x_3, \ldots$, pričom x_k je *najkratší možný* vstup, na ktorom M použije aspoň k políčok.

Na vstupe x_k uvažujme prechodovú postupnosť p, ktorá obsahuje konfiguráciu s k použitými políčkami. Ak je p v prvej polovici, všetky prechodové postupnosti v druhej polovici musia byť rôzne, inak by sme časť vstupu medzi nimi mohli vynechať a dostali by sme kratší vstup na ktorom M použije k políčok. Podobne, ak je p v druhej polovici, potom všetky prechodové postupnosti v prvej polovici musia byť rôzne. Z toho vyplýva, že máme aspoň $n/2$ rôznych prechodových postupností, ale *všetkých* prechodových postupností je najviac $2^{2^{c \cdot s(n)}}$. Z toho vyplýva, že

$$n/2 \leq 2^{2^{c \cdot s(n)}} \quad \Longrightarrow \quad s(n) \geq \tfrac{1}{c} \log\log(n/2) \text{ pre nekonečne veľa } n.$$

Teda ak $s(n) = \omega(1)$, potom nie je $s(n) = o(\log\log n)$. □

Naopak, rozmyslite si, že jazyk

$$L = \{\#\mathrm{bin}_k(0)\#\mathrm{bin}_k(1)\#\cdots\#\mathrm{bin}_k(2^k-1)\# \mid k \geq 0\},$$

kde $\mathrm{bin}_k(x)$ je k-bitový binárny zápis čísla x sa dá akceptovať v $\mathsf{DSPACE}(\log\log n)$ a nie je regulárny.

Dôsledok 3.1. $\mathsf{DSPACE}(\log\log n) \supset \mathsf{DSPACE}(1) = \mathsf{REG}$.

Pri druhom tvrdení treba priznať, že je to tak trochu podvod, alebo, jemnejšie povedané, artefakt nášho modelu. Výpočty Turingovych strojov totiž vieme zrýchliť (lineárne), ale je to na úkor obrovského nárastu počtu stavov, veľkosti abecedy a δ-funkcie. Menšiu konštantu v časovej zložitosti dosiahneme tak, že viacero krokov „zadrátujeme“ do δ-funkcie. Treba tiež povedať, že vo veľa iných modeloch, vrátane RAM, takáto veta neplatí.

Veta 3.2 (Veta o lineárnom zrýchlení). *Pre každý deterministický Turingov stroj M, ktorý počíta v čase $t(n) = \omega(n)$ existuje ekvivalentný stroj M', ktorý počíta v čase $\lceil t(n)/2 \rceil$ (pre dostatočne veľké n).*

■ **Náznak dôkazu.** Stroj M' najskôr „skomprimuje“ vstup: vždy k znakov na vstupe zapíše do jediného políčka na extra páske (M' má oveľa väčšiu abecedu; toto zaberie $2n$ krokov). M' zakaždým načíta políčka, kde sú hlavy + susedné políčka vpravo a vľavo (4 kroky). V tomto momente si M' v jednom stave pamätá k políčok okolo každej hlavy, takže môže odsimulovať aspoň k krokov stroja M *naraz* (toto je zadrátované v δ-funkcii). Následne zapíše zmenené políčka späť na pásku (ďalšie 4 kroky). M' teda pracuje v čase $2n + \lceil 8t(n)/k \rceil$. □

Nakoniec, aj pri tretej vete treba priznať, že ide tak trochu o patológiu a takúto funkciu $t(n)$ v praxi len tak ľahko nestretnete. Ako ukážeme v nasledujúcej kapitole, nie je časovo konštruovateľná, tzn. samotná hodnota $t(n)$ sa nedá vypočítať v čase $t(n)$. Ide o poučný protipríklad, ktorý ukazuje, že požiadavka na časovú (alebo pamäťovú) konštruovateľnosť vo vetách o hierachii naozaj potrebujeme.

Veta 3.3 (Veta o medzere). *Existuje funkcia* $t : \mathbb{N} \to \mathbb{N}$ *taká, že* $\mathsf{DTIME}(t(n)) = \mathsf{DTIME}(2^{t(n)})$.

■ **Dôkaz.** Nech t_i je časová zložitosť i-teho Turingovho stroja M_i. Použijeme metódu diagonalizácie: funkciu $t(n)$ definujeme tak, aby žiadne t_i nebolo medzi $t(n)$ a $2^{t(n)}$, ale robíme to postupne: pri určovaní $t(n)$ obabreme iba stroje od 1 po n, ktorých je iba konečne veľa, ale na každý stroj skôr či neskôr dôjde.

Konkrétne, definujeme

$$\begin{aligned} t(n) &= \text{najmenšie } m \text{ také, že } \forall i \leq n : t_i(n) \leq 2^m \Rightarrow t_i(n) \leq m \\ &= \text{najmenšie } m \text{ také, že } \forall i \leq n : t_i(n) \notin (m, 2^m] \end{aligned}$$

Funkciu $t(n)$ môžeme vypočítať tak, že začneme s $m = 0$ a kým existuje $t_i(n) \in (m, 2^m]$, dosadíme $m \leftarrow t_i(n)$. Keďže pre každé n uvažujeme len konečne veľa strojov M_i, funkcia je dobre definovaná.

Predstavme si, že M_i beží v čase $t_i(n) \leq 2^{t(n)}$. Pre všetky $n \geq i$ je už aj mašina M_i zahrnutá v definícií $t(n)$, z ktorej vyplýva, že ak $t_i(n) \leq 2^{t(n)}$, tak $t_i(n) \leq t(n)$. Teda $t_i(n) = O(t(n))$ a $\mathsf{DTIME}(t(n)) = \mathsf{DTIME}(2^{t(n)})$. □

Táto veta sa dá ďalej zovšeobecniť: Pre ľubovoľnú funkciu $f : \mathbb{N} \to \mathbb{N}$, $f(x) \geq x$ existuje $t(n)$ taká, že problémy riešiteľné v čase $f(t(n))$ sú riešiteľné v čase $t(n)$. Rovnako to platí aj pre pamäťovú zložitosť a dokonca aj pre ľubovoľnú abstraktnú mieru zložitosti (ktorá spĺňa nejaké axiómy, pozri Blum (1967)).

3.2 Hierarchie

Na druhej strane, Áno, ak sa vyhneme patológiam spomínaným vyššie, vo všeobecnosti platí, že ak máme viac času alebo pamäte, vieme vyriešiť viac problémov. Ukážeme si, že to platí pre deterministické aj nedeterministické stroje a tiež pre neuniformné obvody.

Budeme používať Cantorovu techniku diagonalizácie (pozri obrázok 3.2), hoci vo viacerých prípadoch musíme pridať nové finty.

	1	2	3	4	5	6	⋯
S_1	**0**	0	1	0	0	0	⋯
S_2	1	**0**	0	1	1	1	⋯
S_3	0	0	**1**	1	0	0	⋯
S_4	0	1	0	**0**	1	0	⋯
S_5	0	0	0	1	**1**	1	⋯
S_6	0	1	0	1	0	**1**	⋯
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱
$\overline{D}$	1	1	0	1	0	0	⋯

(a) Množín prirodzených čísiel je nespočítateľne veľa ($|\mathbb{N}| \prec |2^{\mathbb{N}}|$). K ľubovoľnej postupnosti množín $S_1, S_2, S_3, \ldots$ (jednotka na pozícii (i,j) znamená, že $j \in S_i$) totiž vieme zostrojiť množinu, ktorá na zozname chýba: stačí zobrať diagonálu a znegovať všetky bity. Zostrojíme tak množinu $\overline{D} = \{i \mid i \notin S_i\}$, ktorá sa líši od všetkých množín na zozname (od S_i líši prvkom i).

	w_1	w_2	w_3	w_4	w_5	w_6	⋯
M_1	**0**	0	1	0	0	0	⋯
M_2	1	**0**	0	1	1	1	⋯
M_3	0	0	**1**	1	0	0	⋯
M_4	0	1	0	**0**	1	0	⋯
M_5	0	0	0	1	**1**	1	⋯
M_6	0	1	0	1	0	**1**	⋯
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱
$\overline{D}$	1	1	0	1	0	0	⋯

(b) Uvažujme enumeráciu všetkých TS $M_1, M_2, M_3, \ldots$ a všetkých slov $w_1, w_2, w_3, \ldots$. Jednotka na pozícií (i,j) v tabuľke znamená, že $w_j \in L(M_i)$. Komplement diagonálneho jazyka $\overline{D} = \{w_i \mid w_i \notin L(M_i)\}$ je nerozhodnuteľný, pretože sa líši od každého rozhodnuteľného jazyka.

Obr. 3.2: Technika diagonalizácie.

Deterministické

Veta 3.4 (Pamäťová hierarchia). *Nech S je páskovo konštruovateľná a $\log n \leq s(n) = o(S(n))$. Potom* $\mathsf{DSPACE}(s(n)) \subset \mathsf{DSPACE}(S(n))$.

Špeciálne $\mathsf{NL} \subset \mathsf{PSPACE} \subset \mathsf{EXPSPACE}$ *a* $\mathsf{DSPACE}(n^\alpha) \subset \mathsf{DSPACE}(n^\beta)$ *pre* $0 \leq \alpha < \beta$.

■ **Dôkaz.** Diagonalizáciou: definujeme Turingov stroj, ktorý pracuje v pamäti $O(S(n))$ a správa sa *inak* ako každý zo strojov s pamäťou $s(n)$. Konkrétne stroj M obabreme na vstupe $w = 1^k\#\langle M\rangle$ (dĺžky n). Budeme simulovať výpočet stroja M na vstupe w a nakoniec rozhodneme *opačne* ako M. Ak akceptoval, my odmietneme a ak odmietol, my akceptujeme. Na páske si vyznačíme $S(n)$ políčok – ak by chcel M použiť viac, môžeme skončiť s ľubovoľnou odpoveďou (buď bolo príliš n malé, alebo stroj M nepracuje v pamäti $s(n)$, takže nás nezaujíma). Okrem toho pri simulácii meriame čas (počet krokov) a ak M počíta dlhšie ako $2^{S(n)}$ krokov, akceptujeme. Pre dostatočne veľké n je totiž $2^{S(n)} > 2^{O(s(n))}$ a teda M buď nepracuje v pamäti $s(n)$, alebo sa zacyklil a my odpovedáme *opačne*. □

Veta 3.5 (Časová hierarchia, Hartmanis a Stearns (1965)). *Nech T je časovo konštruovateľná funkcia, $t(n) \geq n$ a $t(n)\log t(n) = o(T(n))$. Potom platí* $\mathsf{DTIME}(t(n)) \subset \mathsf{DTIME}(T(n))$. *Napríklad* $\mathsf{P} \subset \mathsf{EXP}$.

■ **Dôkaz.** Dôkaz je podobný, s tým rozdielom, že za simuláciu platíme logaritmický čas navyše (pozri kapitolu 2.1). □

Treba si uvedomiť, že táto logaritmická penalta za simuláciu závisí od zvoleného modelu: Turingov stroj s konštantým, no ľubovoľne veľkým počtom pracovných pások. Ak chceme napríklad 100-páskový stroj simulovať s dvoma páskami, platíme logaritmus navyše. Na druhej strane, napríklad Fürer (1982) dokázal, že $\mathsf{DTIME}_{k\text{-TS}}(t(n)) \subset \mathsf{DTIME}_{k\text{-TS}}(T(n))$ pre $n < t(n) = o(T(n))$, ak uvažujeme Turingove stroje s fixným počtom pások $k \geq 2$. Podobne pre RAM s jednotkovou alebo logaritmickou cenou je $\mathsf{DTIME}_{\mathrm{RAM}}(t(n)) \subset \mathsf{DTIME}_{\mathrm{RAM}}(T(n))$ pre $t(n) = o(T(n))$, $T(n) \geq n\ell(n)$, pretože RAMy sa vedia navzájom simulovať len s lineárnym spomalením (Cook a Reckhow, 1973).

Nedeterministické

Veta 3.6 (Nedeterministická pamäťová hierarchia). *Nech* $\log n \leq s(n) = o(S(n))$, *kde* S *je páskovo konštruovateľná. Potom* $\mathsf{NSPACE}(s(n)) \subset \mathsf{NSPACE}(S(n))$.

Teda $\mathsf{NL} \subset \mathsf{NSPACE}(n) \subset \mathsf{PSPACE}$. *(Nedeterministický lineárny priestor sú kontextové jazyky.)*

■ **Dôkaz.** Využijeme, že nedeterministický priestor je uzavretý na komplement. □

Veta 3.7 (Nedeterministická časová hierarchia, Cook (1973), Žák (1983)). *Nech* $t(n+1) = o(T(n))$, *kde* t, T *sú ľubovoľné časovo konštruovateľné funkcie. Potom* $\mathsf{NTIME}(t(n)) \subset \mathsf{NTIME}(T(n))$. *Teda napríklad* $\mathsf{NP} \subset \mathsf{NEXP}$.

■ **Dôkaz.** Nech $n_0 = 1$ a $n_k = 2^{t(n_{k-1})^2}$ a nech i_k je také číslo, že 2^{i_k} je najvyššia mocnina 2 deliaca k (hodnoty i_k tvoria tzv. pravítkovú postupnosť: 0, 1, 0, 2, 0, 1, 0, 3, 0,... každé prirodzené číslo sa v nej nachádza nekonečne veľakrát). Stroj M_{i_k} budeme diagonalizovať na vstupoch dĺžky $n_{k-1}, \ldots, n_k - 1$.

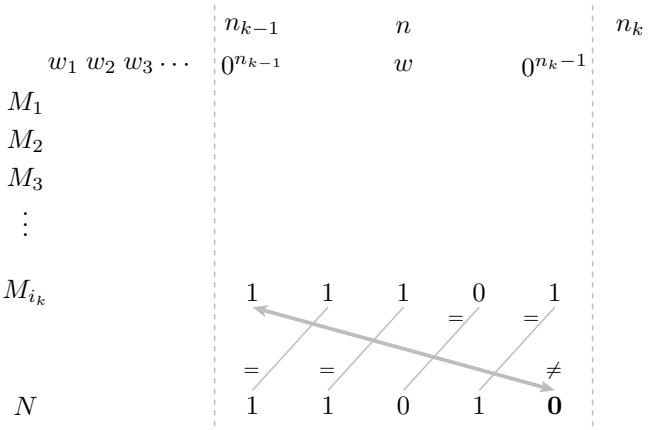
Definujme stroj N takto: Na vstupe 0^n najskôr nájdeme hodnotu k. Finta (pozri obr. 3.3):

1. Pre $n < n_k - 1$ budeme simulovať M_{i_k} na 0^{n+1} (teda na vstupe *o 1 dlhšom!*) $T(n)$ krokov a odpovieme *rovnako* ako M_{i_k}.
2. Ak však $n = n_k - 1$, budeme simulovať M_{i_k} na vstupe $0^{n_{k-1}}$(!) Deterministicky vyskúšame všetky možnosti a odpovieme opačne ako M_{i_k}. Máme dosť času, pretože počítame na kratučkom vstupe: výpočet trvá $2^{t(n_{k-1})} \cdot \mathsf{poly}(t(n_{k-1})) \leq n_k - 1 \leq T(n)$ (pre dosť veľké n).

Ukážme, že M_{i_k} nerozozná takýto jazyk L v čase $O(t(n))$:

- ak pre nejaké $n_{k-1} \leq n < n_k - 1$ $M_{i_k}(0^n) \neq M_{i_k}(0^{n+1})$, tak keďže $N(0^n) = M_{i_k}(0^{n+1})$, $N(0^n) \neq M_{i_k}(0^n)$;
- ak M_{i_k} všetky reťazce 0^n pre $n_{k-1} \leq n < n_k$ akceptuje, alebo všetky odmietne, teda špeciálne $M_{i_k}(0^{n_k-1}) = M_{i_k}(0^{n_k-1})$; avšak z toho, ako sme definovali N je $N(0^{n_k-1}) \neq M_{i_k}(0^{n_k-1})$ a teda $N(0^{n_k-1}) \neq M_{i_k}(0^{n_k-1})$.

□



Obr. 3.3: Na vstupe dĺžky n najskôr nájdeme k také, že $n \in [n_{k-1}, n_k)$. Na vstupoch dĺžky $[n_{k-1}, n_k)$ sa stroj N snaží oklamať M_{i_k}. Pre vstupy kratšie ako $n_k - 1$ rozhodneme *rovnako* M_{i_k} na vstupe o jedna dlhšom; pre vstup dĺžky $n_k - 1$ (na konci intervalu) rozhodneme *opačne* ako M_{i_k} na vstupe dĺžky n_{k-1} (zo začiatku intervalu). Takto M_{i_k} obabreme – M_{i_k} nemôže akceptovať ten istý jazyk ako N, pretože potom by v každom stĺpci museli mať oba stroje rovnakú hodnotu. Z tranzitivity by potom vyplynulo, že $M_{i_k}(w) = N(w')$ pre všetky vstupy w, w' dĺžky $[n_{k-1}, n_k)$. Avšak N sme skonštruovali tak, že $N(0^{n_k-1}) \neq M_{i_k}(0^{n_{k-1}})$.

Neuniformná

Po Turingovych strojoch sa poďme pozrieť na obvody: Je pravda, že väčšími obvodmi vieme aj viac vypočítať?

Odpoveď je ÁNO: V predchádzajúcej kapitole sme spomínali, že každá funkcia sa dá vypočítať obvodom veľkosti $O(n2^n)$. Na druhej strane, existujú funkcie, ktoré sa *nedajú* vypočítať povedzme obvodom s menej ako $2^{n/2}$ hradlami. Prečo? Jednoducho preto, že takých obvodov je málo! Všetkých funkcií je viac ako počet všetkých takých obvodov, takže na nejakú funkciu nám nevyjde obvod (ani keby každý obvod počítal inú funkciu). Toto je takzvaný *počítací argument*: Všetkých funkcií na n bitoch je 2^{2^n}, zatiaľčo obvody veľkosti s vieme zapísať pomocou s^2 bitov (toto je veľmi voľný odhad, o chvíľu ho zlepšíme) a teda existuje najviac $2^{(s^2)}$ takých obvodov a pre $s < 2^{n/2}$ je to menej ako 2^{2^n}.

Takže sme dokázali, že existuje funkcia (označme ju F), ktorá sa nedá vypočítať obvodom veľkosti $2^{n/2}$, ale dá sa vypočítať obvodom veľkosti $O(n2^n)$ (lebo každá sa dá). Vo všeobecnosti môžeme túto medzeru „posunúť“ technikou *paddingu*: napríklad funkcia, ktorá počíta F iba na prvých $n' = 2k \log n$ bitoch (a zvyšok vstupu je balast), sa nedá vypočítať obvodom s menej ako $2^{n'/2} = n^k$ hradlami, ale dá sa vypočítať obvodom s $O(n'2^{n'}) = O(n^{2k} \log n)$ hradlami. Dostávame tak hierarchiu ťažších a ťažších problémov.

Poďme teraz jednotlivé medzivýsledky zlepšiť, spresniť a zmenšiť medzeru medzi možným a nemožným. Ukážeme, že obvody veľkosti s vieme zakódovať pomocou $s \log s + O(s)$ bitov a tým pádom existuje funkcia, ktorá sa nedá vypočítať ani obvodom veľkosti $2^n/n$. Ukážeme tiež, že každá funkcia sa dá vypočítať s $5 \cdot 2^n/n$ hradlami – len 5-krát väčším obvodom (najlepší známy výsledok je ešte tesnejší). Tak dokážeme, že pre každé $s(n) < 2^n/n$ existuje funkcia, ktorá sa nedá vypočítať obvodom veľkosti $s(n)$, ale dá sa vypočítať obvodom veľkosti $5s(n)$.

Veta 3.8 (Shannon (1949)). *Existuje funkcia na n bitoch, ktorá sa nedá vypočítať obvodom veľkosti $s = 2^n/n$ pre dostatočne veľké n.*

■ **Dôkaz.** Počítací argument: počet obvodov veľkosti s je *menší* ako počet všetkých funkcií, takže existuje funkcia, ktorá nemá obvod veľkosti s.

Všetkých funkcií je 2^{2^n} – každú funkciu vieme zapísať tabuľkou 2^n výsledkov pre každý vstup. Na druhej strane obvod veľkosti s vieme zapísať pomocou $m = s \times (2 \log(n + s) + 2) < 2s \log s + O(s)$ bitov: máme s hradiel, pre každé zapíšeme čísla dvoch vstupov (hradlo alebo bit zo vstupu), plus dva bity pre typ hradla. Obvodov je 2^m, čo je menej ako 2^{2^n} už pre $s = 2^n/3n$.

Ešte lepší odhad: Pri zapisovaní obvodov sme nešpecifikovali poradie hradiel – každý obvod môžeme zapísať aspoň $s!$ spôsobmi, takže počet obvodov je v skutočnosti $\leq 2^m/s!$. Inými slovami, pri zápise vieme ušetriť $\log s! = s \log s + O(s)$ bitov (Stirlingova aproximácia) a zakódovať obvody $s \log s + O(s)$ bitmi. Pre $s = 2^n/n$ je to $2^n/n(n - \log n) + O(2^n/n) = 2^n[1 - \log n/n + O(1/n)] < 2^n$. □

Z dôkazu tiež vidieť, že ak s zmenšíme o ε, dĺžka zápisu m sa zmenší aspoň o ε že napríklad polovica funkcií potrebuje obvod veľkosti $2^n/n - 1$ a menej ako 1‰ funkcií sa dá spočítať obvodom veľkosti $2^n/n - 10$. Len menej ako $1/n$-tina všetkých funkcií má obvod veľkosti $2^n/n - \log n$ a exponenciálne málo funkcií má obvod veľkosti $2^n/n - n$. Skrátka – skoro všetky funkcie sú veľmi veľmi ťažké.

Veta 3.9 (Lupanov (1958)). *Každá funkcia na n bitoch sa dá vypočítať obvodom veľkosti $O(2^n/n)$, presnejšie $5 \cdot 2^n/n$.*

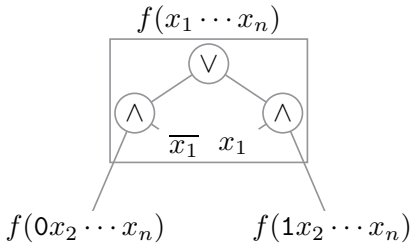
■ **Dôkaz.** V predchádzajúcej kapitole sme spomínali, že každá funkcia sa dá vypočítať obvodom veľkosti $O(n2^n)$.

Lepšie: každá booleovská funkcia n premenných sa dá vypočítať obvodom veľkosti $1.5 \cdot 2^n + n$: stačí ju napísať v tvare

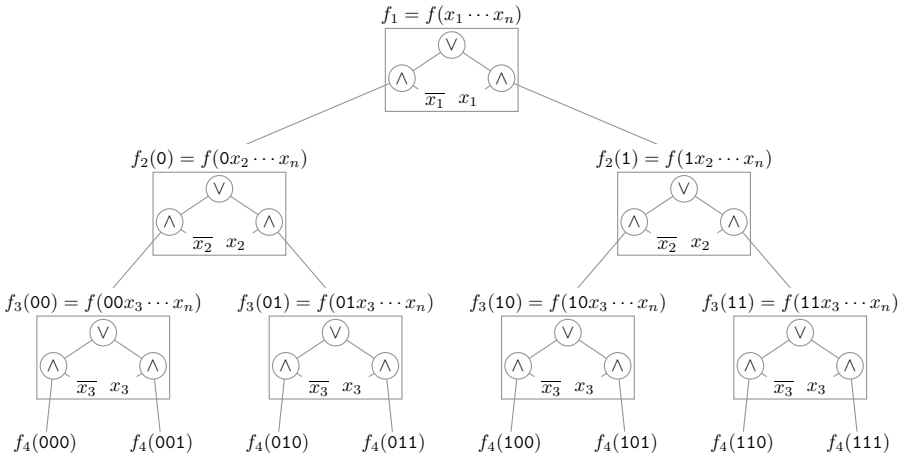
$$f(x_1, \ldots, x_n) = [x_1 \wedge f(1, x_2, \ldots, x_n)] \vee [\overline{x_1} \wedge f(0, x_2, \ldots, x_n)]$$

a rekurzívne vyjadriť dve funkcie $n - 1$ premenných (pozri obrázok 3.4). Toto riešenie si môžeme predstaviť ako rozhodovací strom: ak $x =$ false, výsledok záleží od ľavej vetvy, ak $x =$ true, ideme doprava. Na konci sa dostaneme k zadrátovanej hodnote funkcie na konkrétnom vstupe.

Ak nepočítame negácie (všetkých negácií je dokopy iba n), počet hradiel je $S_n = 3 \times (2^n - 1)$ (úplný strom hĺbky n krát 3 hradlá v každom rámiku).



(a) Rozdelíme vstupy na tie, ktoré začínajú `0` a tie, čo začínajú `1`. Rekurzívne pokračujeme funkciou, ktorá má zafixovaný prvý bit.

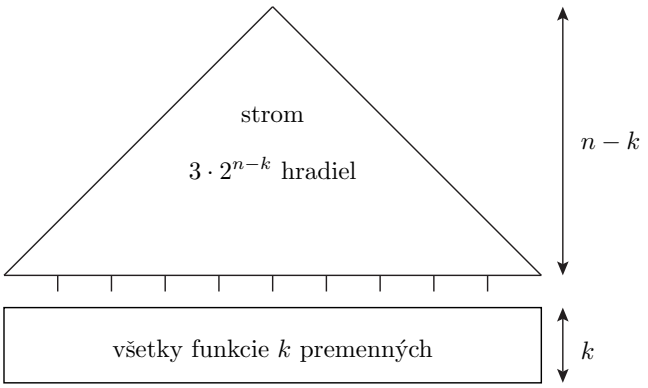


(b) Príklad pre funkciu na 3-och bitoch. V listoch sú konštantné funkcie $f(x)$ pre fixné x, ktoré môžeme zadrátovať do obvodu. Hodnota $\overline{x_3}$ je na obrázku 4×, v praxi, samozrejme, negáciu spočítame len raz a výslednú hodnotu dovedieme drátmi ku všetkým hradlám, kde treba.

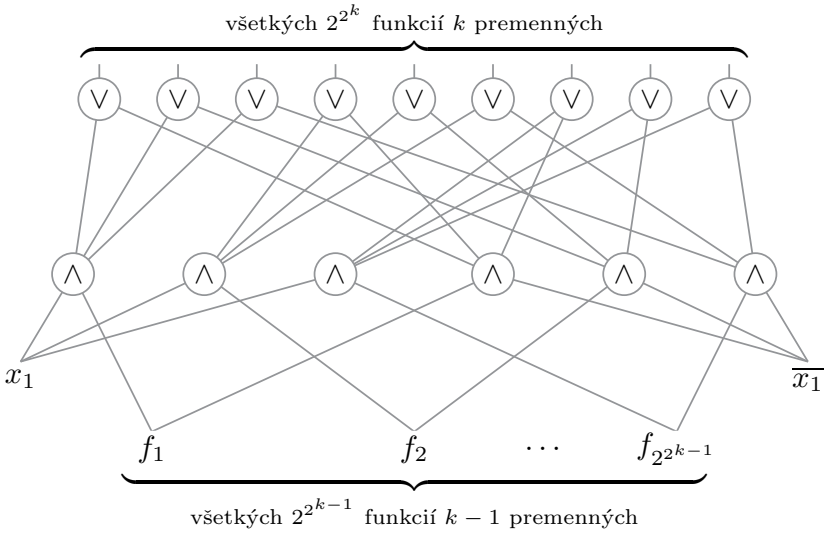
Obr. 3.4: Jednoduchý spôsob, ako skonštruovať obvod veľkosti $O(2^n)$ pre ľubovoľnú funkciu.

Konštrukciu vieme ešte trochu vylepšiť, ak rekurziu zastavíme skôr, napríklad posledná úroveň na obr. 3.4b je zbytočná: funkcie jedinej premennej sú len 0, 1, x a $\overline{x}$. Tá jedna úroveň nám ušetrí polovicu hradiel.

Dá sa to ešte lepšie? Áno, môžeme pokračovať a skúsiť zjednodušiť predposlednú vrstvu: tá zaberá až $3 \times 2^{n-2}$ hradiel, hoci v skutočnosti tam počítame iba funkcie dvoch premenných a je $2^4 = 16$ takých funkcií. Mohli by sme ich teda vypočítať všetky, raz a navždy, namiesto toho, aby sme ich opakovane počítali v každej z 2^{n-2} vetiev.



(a) Pri konštrukcií ako na obrázku 3.4 dostaneme obvod veľkosti zhruba $3 \cdot 2^{n-k}$. Každá ďalšia vrstva pridá dvakrár toľko hradiel, čo predošlá. Ak sa však pozrieme na spodok stromu (spodných k úrovní), zistíme, že je tam zhruba 2^{n-k} vetiev, ale všetko sú to funkcie k premenných, ktorých je 2^{2^k}. Pre veľké n a relatívne malé k sa nám budú funkcie opakovať – obvod bude obsahovať viacero kópií celých podstromov výšky k. Tejto „copy-pasty“ sa chceme zbaviť. V najhoršom prípade sa na spodných k úrovniach vyskytnú všetky funkcie k premenných, avšak stále môže byť výhodnejšie spočítať ich všetky, ako ďalej vetviť a zdvojnásobovať počet hradiel. Samozrejme, funkcia 2^{2^k} rastie veľmi rýchlo, až dvojito exponenciálne, takže existuje nejaká hranica, pokiaľ sa tento prístup oplatí a od určitého k by už začal člen 2^{2^k} dominovať.



(b) Ako spomíname v dôkaze, každá funkcia k premenných sa dá zapísať v tvare $f(x_1, \ldots, x_k) = [x_1 \wedge \underbrace{f(1, x_2, \ldots, x_k)}_{f_1}] \vee [\overline{x_1} \wedge \underbrace{f(0, x_2, \ldots, x_k)}_{f_2}]$, kde f_1 a f_2 sú funkcie $k-1$ premenných. Ak sme teda už zostrojili všetkých $2^{2^{k-1}}$ funkcií $k-1$ premenných, zostrojíme všetky ANDy $g_i = x_1 \wedge f_i$ a $g'_i = \overline{x_1} \wedge f_i$ a následne všetky ORy tvaru $g_i \vee g'_j$, dostaneme všetky funkcie k premenných.

Obr. 3.5: Konštrukcia všetkých funkcií k premenných

Ešte lepšie? Určite. Postup môžeme zovšeobecniť: Všimnime si spodok stromu na obr. 3.4b. Ak obvod usekneme na k-tej úrovni odspodu, hore nám ostane strom s približne $3 \cdot 2^{n-k}$ hradlami a dolu nám ostane zhruba 2^{n-k} vetiev, každá vedie k nejakej funkcii k premenných (pozri obr. 3.5a). Všetkých funkcií k premenných je až 2^{2^k}, ale pre malé k to môže byť stále lepšie ako 2^{n-k}-krát počítať funkciu obvodom veľkosti S_k, na čo treba $O(2^n)$ hradiel. Idea je, že všetky funkcie k premenných spočítame raz a navždy a výsledok správnej funkcie vždy napojíme na hradlá, kde treba.

Ako skonštruujeme všetky funkcie k premenných? Podobne ako v riešení vyššie: Každú funkciu k premenných vieme zapísať ako

$$f(x_1, \ldots, x_k) = [x_1 \wedge \underbrace{f(1, x_2, \ldots, x_k)}_{f_1}] \vee [\overline{x_1} \wedge \underbrace{f(0, x_2, \ldots, x_k)}_{f_2}],$$

kde f_1 a f_2 sú funkcie $k-1$ premenných. Najskôr teda rekurzívne zostrojíme všetky funkcie $k-1$ premenných a následne vytvoríme všetky funkcie tvaru $[x_1 \wedge f_i] \vee [\overline{x_1} \wedge f_j]$, viď obrázok 3.5b. Označme A_k počet hradiel (okrem negácií), ktoré na to budeme potrebovať. Funkcií $k-1$ premenných je $2^{2^{k-1}}$, každú zANDujeme s x_1 a $\overline{x_1}$ (to je $2 \cdot 2^{2^{k-1}}$ ANDov) a následne každú dvojicu zORujeme (dvojíc je $\left(2^{2^{k-1}}\right)^2 = 2^{2^k}$). Dostávame rekurenciu

$$\begin{aligned} A_k &\leq \left(2^{2^{k-1}}\right)^2 + 2 \cdot 2^{2^{k-1}} + A_{k-1} \\ &= 2^{2^k} + \underbrace{3 \cdot 2^{2^{k-1}} + 3 \cdot 2^{2^{k-2}} + 3 \cdot 2^{2^{k-3}} + \cdots + A_1} \\ &< 2^{2^k} + 3k \cdot 2^{2^{k-1}} \\ &= 2^{2^k}(1 + 3k/2^{2^{k-1}}) \end{aligned}$$

Celkovo náš obvod bude mať $3 \cdot 2^{n-k} + A_k + n$ hradiel. Pre $k = \log(n - \log n)$ dostaneme $S_n = 3 \cdot \frac{2^n}{n - \log n} + \frac{2^n}{n}(1 + o(1)) = (4 + o(1))\frac{2^n}{n}$. □

Najlepšie známe výsledky sú ešte oveľa tesnejšie: Označme $L(n)$ veľkosť obvodu pre *najťažšiu* funkciu na n bitoch – $L(n)$ je najmenšie také číslo, že všetky funkcie sú v $\mathsf{SIZE}(L(n))$. $L(n)$ tiež prezývame Shannonova funkcia. Dá sa ukázať (pozri Lozhkin (1997)), že

$$\begin{aligned} L(n) &\geq \frac{2^n}{n}\left(1 + \frac{\log n - O(1)}{n}\right) \\ &\leq \frac{2^n}{n}\left(1 + \frac{\log n + \log\log n + O(1)}{n}\right) \end{aligned}$$

Ba čo viac, ak definujeme $L(n, \varepsilon)$ ako najmenšie s také, že každá funkcia na n bitoch sa dá spočítať obvodom veľkosti s na aspoň $(1/2 + \varepsilon)$-tine vstupov, potom (Andreev a spol., 1997)

$$L(n, \varepsilon) = \Theta\left(\frac{2^n \varepsilon^2}{\log(2 + 2^n \varepsilon^2)}\right) + \Theta(n).$$

Napríklad ak nám stačí spočítať funkciu na 99% vstupov, najťažšie funkcie potrebujú obvod veľkosti $L(n, 0.49) = \Theta(2^n/n)$. Zjavne každú funkciu vieme spočítať na polovici vstupov (buď aspoň polovica vstupov je 1 alebo aspoň polovica je 0). Ak však chceme spočítať funkciu na čo i len $1/2+1/2^{n/10}$ vstupoch, teda len o chlp viac ako na polovic vstupov (pričom ten chlp je exponenciálne malý), v najhoršom prípade potrebujeme $L(n, 2^{-n/10}) = \Theta(2^{4n/5}/n)$, teda exponenciálne veľké obvody.

Veta 3.10 (Neuniformná hierarchia). *Pre každé $s : \mathbb{N} \to \mathbb{N}$, $s(n) < 2^n/n$ je* $\mathsf{SIZE}(s(n)) \subset \mathsf{SIZE}(5s(n))$.

■ **Dôkaz.** Pre každé ℓ existuje funkcia $f : \{0,1\}^\ell \to \{0,1\}$, ktorá sa nedá vypočítať obvodom veľkosti $2^\ell/\ell$, ale každá sa dá vypočítať obvodom veľkosti $5 \cdot 2^\ell/\ell$. Nech $g : \{0,1\}^n \to \{0,1\}$ je funkcia, ktorá aplikuje f na prvých ℓ vstupov, kde $2^\ell/\ell = s(n)$, potom $g \in \mathsf{SIZE}(5s(n))$, ale $g \notin \mathsf{SIZE}(s(n))$. □

Pravdepodobnostná

Platí, že ak máme viac času, vieme riešiť viac problémov aj pri pravdepodobnostných algoritmoch? Intuícia vraví, že asi áno, ale v súčasnosti to nevieme dokázať. Teda až na pomerne trápne výsledky ako napríklad, že keď máme *exponenciálne viac* času, tak dokážeme viac vypočítať:

Veta 3.11. $\mathsf{BPP} \subset \mathsf{BPEXP}$.

■ **Dôkaz.** Označme EEXP a BPEEXP triedu problémov riešiteľných deterministicky, respektíve pravdepodobnostne v dvojito-exponenciálnom čase, t.j. v čase $2^{2^{\mathsf{poly}(n)}}$. Vieme, že $\mathsf{BPP} \subseteq \mathsf{EXP} \subset \mathsf{EEXP} \subseteq \mathsf{BPEEXP}$ (deterministická hierarchia), teda triviálne $\mathsf{BPP} \subset \mathsf{BPEEXP}$ – ešte trápnejší výsledok.

Avšak keby $\mathsf{BPP} = \mathsf{BPEXP}$, potom technikou paddingu dostaneme, že $\mathsf{BPEXP} = \mathsf{BPEEXP}$ (ak $L \in \mathsf{BPTIME}(2^{2^{n^k}})$, vytvorme $L' = \{x\#0^{2^{|x|^k}} \mid x \in L\}$, ten sa dá riešiť v $\mathsf{BPTIME}(2^n)$ – vzhľadom na väčšiu dĺžku vstupu; ale z predpokladu potom $L' \in \mathsf{BPP}$, takže $L \in \mathsf{BPEXP}$), takže keby $\mathsf{BPP} = \mathsf{BPEXP}$, potom $\mathsf{BPP} = \mathsf{BPEEXP}$ a to je spor. □

Ale dá sa napríklad v kvadratickom čase vypočítať viac, ako v lineárnom? To zatiaľ nevieme dokázať.

V čom je problém? Skúste sa zamyslieť sami – skúsim tu odprezentovať „akože-vetu“ a jej chybný(!) „akože-dôkaz“. Nájdite v ňom chybu:

Neplatná veta 1. *Nech $n \leq t(n) = o(T(n)/\log t(n))$, kde T je časovo konštruovateľná. Potom* $\mathsf{BPTIME}(t(n)) \subset \mathsf{BPTIME}(T(n))$.

■ **Zlý dôkaz.** Diagonalizáciou: definujeme pravdepodobnostný Turingov stroj, ktorý pre každý stroj pracujúci v čase $t(n)$ dá na aspoň jednom slove iný výsledok. Konkrétne stroj M obabreme na vstupe $w = 1^k\#\langle M\rangle$ dĺžky n: simulujeme M $T(n)$ krokov (hádžeme mincou tak, ako M) a rozhodneme *opačne*

ako M. Za simuláciu k-páskového stroja dvoma páskami zaplatíme logaritmus navyše, ale pre dosť veľké n je $c \cdot t(n) \log t(n) < T(n)$. □

Úlohy

- Aký je problém v dôkaze pravdepodobnostnej hierarchie v „ne-vete“ 1?
- Uveďte príklad funkcie, ktorá rastie asymptoticky rýchlejšie ako polynomialne, ale pomalšie ako exponenciálne. (Tzn. pre ľubovoľné $k \in \mathbb{N}$ a $c > 0$ od nejakého n_0 platí: $n^k < f(n) < c^n$.)
- Dokážte vetu 3.6.
- Dokážte, že existuje jazyk $L \in \mathsf{EXPSPACE}$, ktorý nepatrí do $\mathsf{SIZE}(2^n/n)$.
- Dokážte, že pre ľubovoľne veľké fixné $k \in \mathbb{N}$ existuje jazyk $L \in \mathsf{EXP}$ taký, že $L \notin \mathsf{P}/n^k$. Teda dá sa rozhodovať v exponenciálnom čase, ale nedá sa v polynomiálnom čase, ani s radou dĺžky n^k. (Hint: Použite diagonalizáciu.)

Literatúra

Andreev, Alexander E, Andrea EF Clementi, a José DP Rolim. 1997. “Optimal bounds for the approximation of boolean functions and some applications.” *Theoretical Computer Science* 180(1-2), s. 243–268.

Blum, Manuel. 1967. “A machine-independent theory of the complexity of recursive functions.” *Journal of the ACM (JACM)* 14(2), s. 322–336.

Cook, Stephen A. 1973. “A hierarchy for nondeterministic time complexity.” *Journal of Computer and System Sciences* 7(4), s. 343–353.

Cook, Stephen A a Robert A Reckhow. 1973. “Time bounded random access machines.” *Journal of Computer and System Sciences* 7(4), s. 354–375.

Fürer, Martin. 1982. “The tight deterministic time hierarchy.” In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*. s. 8–16.

Hartmanis, Juris a Richard E Stearns. 1965. “On the computational complexity of algorithms.” *Transactions of the American Mathematical Society* 117, s. 285–306.

Lozhkin, SA. 1997. “Asymptotic bounds of high accuracy on the complexity of control systems.” *Doctor thesis, Moscow State University* .

Lupanov, Oleg Borisovich. 1958. “The synthesis of contact circuits.” In *Doklady Akademii Nauk*, roč. 119. Russian Academy of Sciences, s. 23–26.

Shannon, Claude E. 1949. “The synthesis of two-terminal switching circuits.” *The Bell System Technical Journal* 28(1), s. 59–98.

Žák, Stanislav. 1983. "A Turing machine time hierarchy." *Theoretical Computer Science* 26(3), s. 327–333.

Kapitola 4

Redukcie, ťažkosť, úplnosť

Príklad 1: Čo je ťažšie: násobenie $x \cdot y$, alebo umocňovanie na druhú: x^2?

Príklad 2: Čo je ťažšie: utriediť n čísel alebo spočítať konvexný obal množiny bodov?

Príklad 3: Čo je ťažšie: ofarbiť planárny graf 3 farbami (pričom susedné vrcholy musia mať rôznu farbu) alebo ofarbiť ľubovoľný graf?

Príklad 4: Čo je ťažšie: zistiť, či sú dva regulárne výrazy ekvivalentné, alebo ako vyhrať Super Maria?

Príklad 5: Čo je ťažšie: zistiť, či daný TS M zastaví na daný počet n krokov, alebo nájsť vyhrávajúcu stratégiu v zovšeobecnenej $n \times n$ dáme?

Na prvý pohľad je umocňovanie na druhú len špeciálny prípad násobenia a ofarbovanie planárnych grafov špeciálny prípad farbernia všeobecných grafov. Všeobecnejší problém by teda mal byť aj ťažší. A na prvý pohľad možno nie je jasné, čo má spoločné triedenie a konvexné obaly, alebo regulárne výrazy a hra Super Mario, či Turingove stroje a dáma.

V skutočnosti sú v každom príklade oba problémy *zhruba* rovnako ťažké. A čo je zaujímavé, vieme to povedať napriek tomu, že vo viacerých prípadoch nepoznáme skutočnú zložitosť jednotlivých problémov. Taká je sila redukcií.

Ak chceme dokázať, že problém A je aspoň tak ťažký ako B, stačí ukázať, že *ak* by sme vedeli rýchlo riešiť A, tak by sme vedeli rýchlo riešiť aj B. Dá sa to dokázať tak, že jeden problém prevedieme na druhý a ukážeme, ako by sme využili riešenie jedného, pri riešení druhého.

Napríklad, ak by sme vedeli rýchlo umocňovať na druhú, tak vynásobiť dve čísla môžeme nasledovne:

$$x \cdot y = \frac{(x+y)^2 - (x-y)^2}{4}.$$

Rozmyslite si, že sčítanie/odčítanie/delenie štyrmi je jednoduché a trvá lineárny čas. Ak by sme vedeli umocňovať na druhú n-bitové čísla v čase $s(n)$, potom by sme vedeli násobiť v čase $2s(n+1) + O(n)$. Najlepší (a predpokladá

sa, že optimálny) algoritmus násobí n-bitové čísla v čase $O(n \log n)$ (Harvey a van der Hoeven, 2019). Ak by sme vedeli umocňovať na druhú *rýchlejšie* ako $n \log n$, tak aj násobiť by sme vedeli rýchlejšie. Naopak, ak je $n \log n$ algoritmus pre násobenie optimálny, tak ani umocňovať na druhú nevieme lepšie ako v $\Omega(n \log n)$.

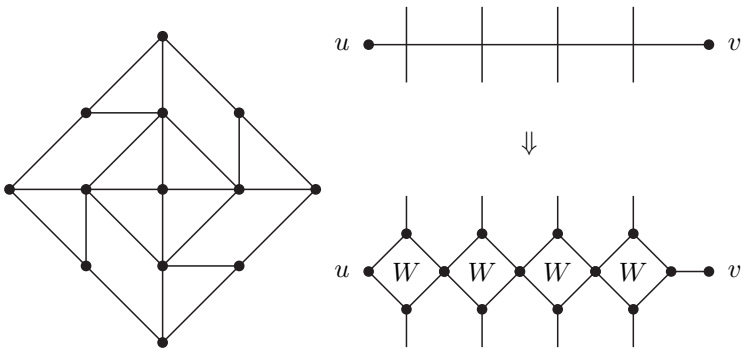
Konvexný obal vieme vypočítať v $O(n \log n)$ napríklad Grahamovym, resp. Andrewovym algoritmom, ktorý body najskôr zotriedi – a triedenie je v skutočnosti tá najpomalšia časť, zvyšok beží v lineárnom čase. Naopak, predstavme si, že chceme zotriediť postupnosť čísel $x_1, x_2, \ldots, x_n$. Vytvorme z nich body (x_1, x_1^2), $(x_2, x_2^2), \ldots, (x_n, x_n^2)$ (teda y-súradnica bude x na druhú). Tieto body ležia na parabole a konvexný obal ich obsahuje všetky, v utriedenom poradí. Ak by sme teda vedeli zostrojiť konvexný obal rýchlejšie ako v $O(n \log n)$, vedeli by sme aj triediť rýchlejšie ako v $O(n \log n)$ a naopak. Ak $t(n)$ je najlepší čas triedenia n prvkov a $k(n)$ je najlepší čas pre nájdenie konvexného obalu, potom

$$t(n) = k(n) \pm O(n).$$

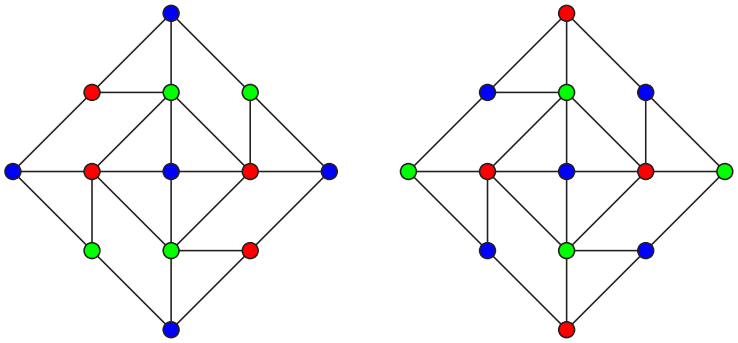
Problém farbenia grafov (Coloring) je NP-úplný pre 3 a viac farieb. Ako je to s planárnymi grafmi (nakreslenými do roviny tak, že sa hrany nekrížia)? Možno ste počuli o vete o štyroch farbách: ak chceme ofarbiť súvislé oblasti na ľubovoľnej mape tak, aby susedné oblasti mali rôznu farbu, stačia nám len štyri farby! Preložené do reči teórie grafov: súvislé oblasti sú vrcholy a susedné oblasti spojíme hranou; dostaneme planárny graf a ten sa *vždy* dá ofarbiť 4 farbami. Takže pre 4 farby je rozhodovací problém farbenia vo všeobecnosti ťažký, ale pre planárne grafy triviálny.

Ako je to však s 3 farbami? Problémy sú (zhruba) rovnako ťažké. Ukážeme si, prečo: Predstavme si, že máme super-rýchly algoritmus, ktorý dokáže ofarbovať 3 farbami v čase $f(n)$, ale iba planárne grafy. Potrebujeme graf, ktorý, žiaľ, nie je planárny. Čo sa dá robiť? Môžeme ho nakresliť do roviny aj tak. Nejaké hrany sa nám budú krížiť, ale to nevadí. Každý jeden priesečník nahradíme planárnym grafom W ako na obrázku 4.1. Výsledný graf bude planárny. Presvedčte sa, že akékoľvek farbenie W musí mať protiľahlé cípy rovnakej farby. Naopak, ofarbenia na obrázku 4.1c ukazujú, že ak pre protiľahlé cípy zvolíme rovnakú farbu, zvyšok grafu už vieme vždy dofarbiť (ak chceme cípy inej farby ako na obrázku, stačí farby spermutovať). Z toho vyplýva, že výsledný graf je 3-ofarbiteľný práve vtedy, keď bol 3-ofarbiteľný pôvodný graf a z riešenia pre jeden graf vieme triviálne získať riešenie pre druhý a naopak.

Na planárny graf, ktorý sme takto zostrojili, by sme mohli použiť náš super-rýchly algoritmus. Aký bude výsledný čas? Nuž, všimnite si, že graf sa nám trochu nafúkne – za každý priesečník nám pribudne 12 vrcholov. Priesečníkov je v najhoršom prípade až $\Theta(n^4)$, takže výsledná zložitosť bude najviac $F(n) = f(12n^4 + n) + O(n^4)$. Nie je to málo, no ak trochu poodstúpime a pozrieme sa na tú zložitosť z diaľky, tak: 1. ak f je polynóm, tak aj F je polynóm; 2. ak f je subexponenciálna funkcia, tak aj F je subexponenciálna; 3. ak sa 3-farbenie grafov nedá lepšie ako v exponenciálnom čase, tak sa to nedá ani pre špeciálny prípad planárnych grafov. (Najlepší známy algoritmus pre 3-farbenie beží v čase



(a) Graf W. (b) Kríženia nahradíme kópiami W.



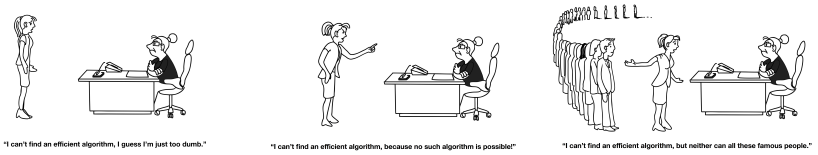
(c) Dve možné ofarbenia grafu W.

Obr. 4.1: Redukcia 3-farbenia grafu na 3-farbanie planárneho grafu.

$O(1.3289^n)$ (Beigel a Eppstein, 2005). Keďže problém je NP-úplný, nevieme, či existuje polynomiálny algoritmus – ak by však existoval, P = NP.)

Super Mario aj ekvivalencia regulárnych výrazov sú tzv. PSPACE-úplné problémy – najťažšie problémy, ktoré sa dajú riešiť v polynomiálnej pamäti. Nevieme ich riešiť v polynomiálnom čase a keby sme to vedeli, potom P = PSPACE (všetky problémy riešiteľné v polynomiálnej pamäti sa dajú riešiť aj v polynomiálnom čase, čo je nepravdepodobné). PSPACE-úplnosť Super Maria si dokážeme v kapitole .

Problém zastavenia na n krokov[1] aj $n \times n$ dáma sú EXP-úplné problémy – najťažšie problémy, ktoré sa dajú riešiť v exponenciálnom čase. Tieto problémy sa *dokázateľne nedajú* riešiť v polynomiálnom čase. O dáme si to ukážeme v kapitole .



(a) Ošemetná situácia: „Neviem nájsť efektívny algoritmus, asi som príliš blbá.“

(b) Ideálny prípad: „Nemôžem nájsť efektívny algoritmus, pretože žiadny taký dokázateľne neexistuje!“ Bohužiaľ, takýchto prípadov je málo.

(c) Realistický scenár: „Neviem nájsť efektívny algoritmus, ale nedokážu to ani títo slávni vedci.“ Stačí problém redukovať na známy ťažký problém.

Obr. 4.2: Vedieť dokázať, že problém je ťažký, je užitočné a praktické. Zdroj: Stefan Szeider, `https://www.ac.tuwien.ac.at/people/szeider/cartoon/`

4.1 Redukcie

Definícia 4.1. *Hovoríme, že problém A je polynomiálne redukovateľný na B, píšeme $A \leq_{\mathrm{m}}^{\mathrm{P}} B$, ak existuje zobrazenie σ vypočítateľné v polynomiálnom čase také, že pre každé x je $x \in A \iff \sigma(x) \in B$. Podobne A je logspace redukovateľný na B, $A \leq_{\mathrm{m}}^{\log} B$, ak sa zobrazenie σ dá vypočítať v logaritmickom priestore. Funkciu σ voláme tiež many-to-one alebo Karpova redukcia A na B.*

Definícia 4.2. *Hovoríme, že L je NP-ťažký, ak $A \leq_{\mathrm{m}}^{\log} L$ pre každý $A \in \mathsf{NP}$. L je NP-úplný, ak je NP-ťažký a patrí do NP. Vo všeobecnosti, ak $\leq$ je redukcia a $\mathcal{C}$ je ľubovoľná trieda problémov, tak hovoríme, že L je $\mathcal{C}$-ťažký (pri $\leq$ redukcii), ak $A \leq L$ pre každé $A \in \mathcal{C}$ a $\mathcal{C}$-úplný, ak L navyše patrí do $\mathcal{C}$.*

Veta 4.1. *Zopár jednoduchých faktov:*

- *Relácie $\leq_{\mathrm{m}}^{\log}, \leq_{\mathrm{m}}^{\mathrm{P}}$ sú tranzitívne: ak $A \leq B$ a $B \leq C$, tak $A \leq C$.*
- *Triedy ako $\mathsf{L}, \mathsf{NL}, \mathsf{P}, \mathsf{NP}, \mathsf{coNP}, \mathsf{PSPACE}, \mathsf{EXP}, \mathsf{NEXP}, \ldots$ sú uzavreté na logspace redukciu: Ak $A \leq_{\mathrm{m}}^{\log} B$ a $B \in \mathcal{C}$, tak $A \in \mathcal{C}$, kde $\mathcal{C}$ je jedna zo spomínaných tried. S výnimkou L a NL, teda od P „vyššie“, sú tieto triedy uzavreté aj na polynomiálnu redukciu.*
- *$A \in \mathsf{P}$ práve vtedy, keď $A \leq_{\mathrm{m}}^{\mathrm{P}} \{1\}$; $A \in \mathsf{L}$ práve vtedy, keď $A \leq_{\mathrm{m}}^{\log} \{1\}$.*
- *Ak $A \leq_{\mathrm{m}}^{\log} B$, tak $A \leq_{\mathrm{m}}^{\mathrm{P}} B$, teda $\leq_{\mathrm{m}}^{\log}$ je zjemnením relácie $\leq_{\mathrm{m}}^{\mathrm{P}}$.*
- *Ak $\mathcal{C}$-úplný jazyk patrí do $\mathcal{C}'$ a $\mathcal{C}'$ je uzavretá na danú redukciu, potom $\mathcal{C} \subseteq \mathcal{C}'$. Napríklad ak NP-úplný jazyk patrí do coNP, potom $\mathsf{NP} = \mathsf{coNP}$. Ak EXP-úplný jazyk patrí do PSPACE, tak $\mathsf{EXP} = \mathsf{PSPACE}$, atď.*

[1] predpokladáme, že číslo n je na vstupe zapísané binárne a teda počet krokov je exponenciálny od dĺžky vstupu

Definícia 4.3. *Hovoríme, že problém A je polynomiálne Turingovsky redukovateľný na B, píšeme $A \leq_{\mathrm{T}}^{\mathrm{P}} B$, ak existuje Turingov stroj M, ktorý používa B ako orákulum (čiernu krabičku) a v polynomiálnom čase rieši A. Takýto stroj má špeciálnu pásku, na ktorú keď napíše vstup x, v konštantnom čase sa dozvie, či $x \in B$. Stroj M s orákulom B zapisujeme M^B, teda $A \leq_{\mathrm{T}}^{\mathrm{P}} B$, ak $A = L(M^B)$, pre polynomiálny stroj M.*

Definícia 4.4. *Nech B je jazyk a $\mathcal{C}, \mathcal{O}$ sú triedy jazykov. Definujeme*

$$\begin{aligned}
\mathsf{P}^B &= \{L(M^B) \mid \textit{kde } M \textit{ je deterministický polynomiálny Turingov stroj}\} \\
\mathsf{NP}^B &= \{L(M^B) \mid \textit{kde } M \textit{ je nedeterministický polynomiálny Turingov stroj}\} \\
\mathsf{P}^{\mathcal{O}} &= \bigcup_{B \in \mathcal{O}} \mathsf{P}^B, \qquad \mathsf{NP}^{\mathcal{O}} = \bigcup_{B \in \mathcal{O}} \mathsf{NP}^B, \qquad \textit{všeobecne } \mathcal{C}^{\mathcal{O}} = \bigcup_{B \in \mathcal{O}} \mathcal{C}^B
\end{aligned}$$

Veta 4.2. *Platí:*

- *Relácia $\leq_{\mathrm{T}}^{\mathrm{P}}$ je tranzitívna.*
- *$A \leq_{\mathrm{T}}^{\mathrm{P}} B$ práve vtedy, keď $A \in \mathsf{P}^B$.*
- *Triedy P a PSPACE sú uzavreté na $\leq_{\mathrm{T}}^{\mathrm{P}}$, teda $\mathsf{P}^{\mathsf{P}} = \mathsf{P}$ a $\mathsf{P}^{\mathsf{PSPACE}} = \mathsf{PSPACE}$.*
- *Relácia $\leq_{\mathrm{m}}^{\mathrm{P}}$ je zjemnením $\leq_{\mathrm{T}}^{\mathrm{P}}$.*

Predpokladá sa, že NP nie je uzavretá na Turingovskú redukciu – potom by totiž $\mathsf{NP} = \mathsf{coNP}$.

4.2 Ťažké a úplné problémy

NP-úplné problémy

Veta 4.3 (Cook (1971), Levin (1973)). *Problém* SAT *je* NP*-úplný.*

■ **Dôkaz.** Chceme dokázať, že každý jazyk $L \in \mathsf{NP}$ vieme redukovať na 3SAT. Nech M je NTS, ktorý ho rozpoznáva v čase n^k. Ukážeme, ako pre každý vstup x zostrojiť formulu ϕ_x takú, že $x \in L \iff \phi_x$ je splniteľná.

Predstavme si akceptačný výpočet M na x: $C_1 \vdash_M C_2 \vdash_M \cdots \vdash_M C_N$ (kde $N = n^k$). Môžeme si ho predstaviť ako tabuľku N krokov $\times$ $(N+2)$ políčok, pričom na každom políčku je nejaký symbol a môže na ňom byť hlava v nejakom stave. Konfigurácie zarovnáme na rovnakú dĺžku a pridáme zarážky $\vdash C \dashv$. Nepoužité políčka budeme značiť „␣“.

Tento výpočet zakódujeme do booleovských premenných a formula ϕ_x bude kontrolovať, či ide o korektný akceptačný výpočet. Teda

$$\begin{aligned}
x \in L &\iff \exists\text{akceptačný výpočet } M \text{ na } x \\
&\iff \exists\text{nastavenie premenných kódujúce tento výpočet.}
\end{aligned}$$

Premenné budú:

- $Q^q_{i,j}$ – v i-tom kroku je hlava na j-tom políčku v stave q a
- $S^a_{i,j}$ – v i-tom kroku je na j-tom políčku symbol a.

Podmienky:

- Na každom políčku je práve jeden symbol:

$$\text{pre každé } i,j: \quad \bigvee_{a\in\Gamma} S^a_{i,j} \qquad \textit{(aspoň jeden)}$$

$$\text{pre každé } i,j \text{ a } a\neq b\in\Gamma: \quad \overline{S^a_{i,j}} \vee \overline{S^b_{i,j}} \qquad \textit{(najviac jeden).}$$

- Podobne hlava je v každom kroku na práve jednom políčku v práve jednom stave:

$$\text{pre každé } i: \quad \bigvee_{\substack{q\in Q\\ 0<j\leq N+1}} Q^q_{i,j} \qquad \textit{(aspoň jeden)}$$

$$\text{pre každé } i, j\neq j', \text{ stavy } p\neq q: \quad \overline{Q^p_{i,j}} \vee \overline{Q^q_{i,j'}} \qquad \textit{(najviac jeden).}$$

- Počiatočná konfigurácia začína v stave q_0 (podľa M):

$$Q^{q_0}_{1,1}$$

a na páske je $\vdash x_1x_2\cdots x_n\text{␣␣␣}\cdots\text{␣}\dashv$ (vstup x doplnený o prázdne políčka na dĺžku N):

$$S^{\vdash}_{1,0}\wedge S^{x_1}_{1,1}\wedge S^{x_2}_{1,2}\wedge\cdots\wedge S^{x_n}_{1,n}\wedge \bigwedge_{n<j\leq N} S^{\text{␣}}_{1,j}\wedge S^{\dashv}_{1,N+1}$$

Toto je jediná časť, ktorá závisí od vstupu x.

- Každá ďalšia konfigurácia musí vždy naväzovať na predošlú a meniť sa podľa δ-funkcie stroja M. Vo všeobecnosti, každé políčko $(i+1,j)$ závisí najviac od troch políčok v predošlej konfigurácii. Ak na políčku j nie je hlava, tak v nasledujúcom kroku bude na políčku rovnaký symbol.

$$(S^a_{i,j}\wedge(\overline{Q^{q_1}_{i,j}}\wedge\cdots\wedge\overline{Q^{q_k}_{i,j}}))\rightarrow S^a_{i+1,j}$$

Ak tam je hlava, symbol sa zmení podľa δ-funkcie stroja M. Konkrétne, ak $\delta(p,a)=(q,b,d)$, tak

$$(S^a_{i,j}\wedge Q^p_{i,j})\rightarrow(S^b_{i+1,j}\wedge Q^q_{i+1,j+d}).$$

Tieto výrazy môžeme prepísať do CNF (podľa $p\rightarrow q\equiv\neg p\vee q$):

$$\text{pre každé } i<N, j, a\in\Gamma \text{ a } ((p,a),(q,b,d))\in\delta:$$

$$(\overline{S^a_{i,j}}\vee Q^{q_1}_{i,j}\vee\cdots\vee Q^{q_m}_{i,j}\vee S^a_{i+1,j})\wedge(\overline{S^a_{i,j}}\vee\overline{Q^p_{i,j}}\vee S^b_{i+1,j})\wedge(\overline{S^a_{i,j}}\vee\overline{Q^p_{i,j}}\vee Q^q_{i+1,j+d}).$$

- Výpočet je akceptačný (môžeme predpokladať, že M sa vráti na prvé políčko a prejde do stavu $ACCEPT$):

$$Q_{N,1}^{ACCEPT}.$$

Všetky tieto klauzuly spojíme $\wedge$-om do jednej formuly ϕ_x, ktorá má $O(N^2)$ premenných a dĺžku $O(N^3) = O(n^{3k}) = \mathsf{poly}(n)$. □

Veta 4.4. *Nasledujúce problémy sú* NP*-úplné:*

- NtmAcc – *dané sú kód NTS M, vstup x a počet krokov t (zapísané unárne); chceme zistiť, či $M(x)$ akceptuje na t krokov; formálne:*
  $\textsc{NtmAcc} = \{\langle M\rangle\#x\#1^t \mid M \text{ je NTS, ktorý akceptuje } x \text{ na } t \text{ krokov}\}$,

- TmSat – *dané sú kód DTS M, dĺžka n a počet krokov t (zapísané unárne); chceme zistiť, či existuje vstup x dĺžky n, ktorý M akceptuje na t krokov; formálne:*
  $\textsc{TmSat} = \{\langle M\rangle\#1^n\#1^t \mid M \text{ je DTS} \wedge \exists\pi \in \{0,1\}^n : M(\pi) \text{ akceptuje na } t \text{ krokov}\}$,

- CircuitSat – *daný je booleovský obvod C; existuje vstup, ktorý akceptuje?*

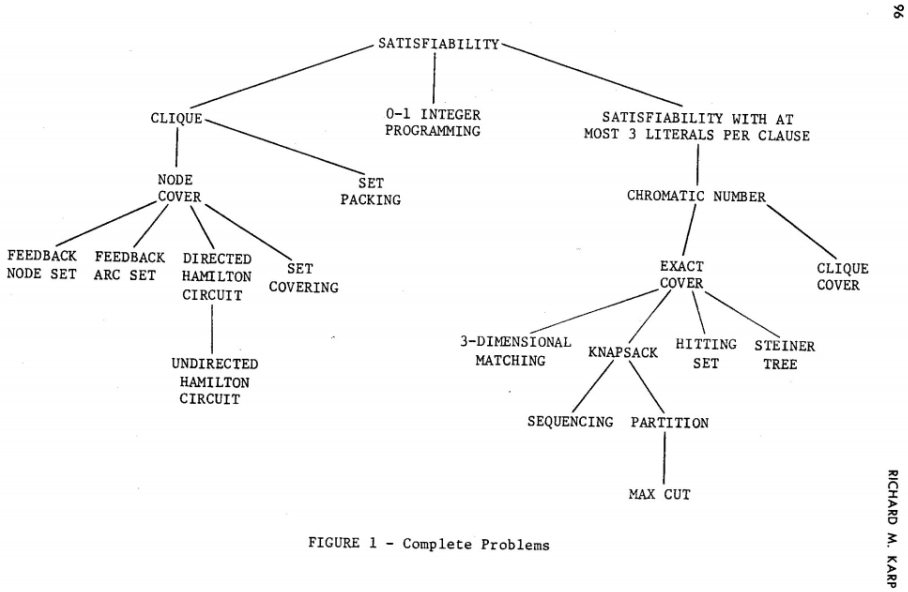
- Sat – *je daná formula ϕ v CNF*[2] *splniteľná? existuje pravdivostné ohodnotenie premenných, pri ktorých je formula pravdivá?*

■ **Dôkaz.** To, že TmSat je NP-úplný, je jasné z definície NP cez certifikáty: $L \in \mathsf{NP}$, ak existuje deterministický stroj M, ktorý v polynomiálnom čase overuje certifikáty a $x \in L \iff \exists\pi : M(x,\pi)$. Pri redukcii stačí „zadrátovať“ vstup x do M a vypísať potrebný počet jednotiek.

$\textsc{TmSat} \leq_{\mathrm{m}}^{\mathrm{P}} \textsc{CircuitSat}$, pretože DTS, ktoré pracujú v polynomiálnom čase sú ekvivalentné polynomiálne veľkým obvodom. Dôkaz tohto tvrdenia je podobný dôkazu Cook-Levinovej vety vyššie – zostrojíme obvod, ktorý dokáže počítať nasledujúcu konfiguráciu $C \vdash_M C'$. Ak však trochu abstrahujeme od detailov, tak každé políčko v ďalšej konfigurácii závisí len od troch políčok v predošlej konfigurácii – a teda od konštantne veľa vstupov a pre každú funkciu k vstupov vieme vyrobiť triviálny CNF obvod veľkosti $O(k2^k) = O(1)$.

Nakoniec $\textsc{CircuitSat} \leq_{\mathrm{m}}^{\mathrm{P}} \textsc{Sat}$, pretože každý obvod C vieme prerobiť na CNF formulu, ak pridáme ďalšie premenné. Pre každé hradlo h so vstupmi f, g budeme mať podľa typu hradla klauzuly $(h \leftrightarrow f \wedge g)$ alebo $(h \leftrightarrow f \vee g)$ alebo $(h \leftrightarrow \neg f)$, prepísané do CNF. Zatiaľčo v CircuitSat sa pýtame, či existuje vstup, na ktorom dá obvod odpoveď 1, v Sat inštancii sa budeme pýtať, či existuje vstup a nastavenie pomocných premenných pre výstupy hradiel a formula už len skontroluje, že pre každé hradlo vstupy a výstup zodpovedajú. Podobne ako v predošlom dôkaze teda nechávame nedeterminizmus, aby natipoval nielen vstupy, ale celý priebeh výpočtu a potom už len skontrolujeme, či lokálne všetko sedí. □

[2] v konjunktívnom normálnom tvare



Obr. 4.3: 21 Karpových problémov a redukcie medzi nimi. Všetky tieto problémy sú NP-úplné.

Veta 4.5 (Karp (1972)). *Nasledujúce problémy sú* NP*-úplné:*

- Sat, 3Sat – *je daná formula v CNF, resp. v 3CNF splniteľná?*
- 0-1-ILP – *pre celočíselnú maticu A a vektor b, existuje 0-1-vektor x taký, že $Ax = b$?*
- Clique – *obsahuje daný graf k-kliku, t.j. k navzájom prepojených vrcholov?*
- SetPacking – *dá sa z množín $S_1, \ldots, S_n$ vybrať k disjunktných množín?*
- NodeCover – *má graf vrcholové pokrytie veľkosti najviac k, t.j. existuje množina vrcholov C taká, že každá hrana má aspoň jeden koniec v C?*
- FeedbackNodeSet – *má daný orientovaný graf G množinu k vrcholov F takú, že každý cyklus v G ide cez nejaký vrchol z F?*
- FeedbackArcSet – *množinu k hrán takú, že každý cyklus v G použije nejakú hranu z F?*
- DirectedHamiltonianCycle *a* UndirectedHamiltonianCycle – *existuje v grafe cyklus, ktorý prechádza cez všetky vrcholy?*
- ChromaticNumber *(k-*Coloring*)* – *dá sa graf ofarbiť k farbami (susedné vrcholy musia mať rôznu farbu)?*

- CliqueCover – *je daný graf zjednotením najviac k klík?*
- SetCovering – *dá sa z množín $S_1, \ldots, S_n$ vybrať k množín, ktoré „pokryjú“ (ich zjednotenie je) celé $\bigcup_{i=1}^{n} S_i$?*
- ExactCover – *dajú sa z množín $S_1, \ldots, S_n$ vybrať disjunktné množiny, ktoré pokryjú celé $\bigcup_{i=1}^{n} S_i$?*
- HittingSet – *existuje pre dané množiny $S_1, \ldots, S_n$ množina H taká, že $|H \cap S_i| = 1$ pre každé i?*
- SteinerTree – *dá sa množina vrcholov v ohodnotenom grafe pospájať hranami s celkovou dĺžkou najviac k?*
- 3dMatching – *máme hypergraf, kde každá hrana spája 3 vrcholy ($E \subseteq V \times V \times V$);*
- Knapsack – *dané celé čísla $a_1, \ldots, a_n, b$; dá sa vybrať podmnožina a-čok so súčtom b?*
- JobSequencing – *máme n úloh, pre každú máme čas trvania, deadline, dokedy ju treba stihnúť a penále, ktoré musíme zaplatiť, ak to nestihneme; dajú sa úlohy zoradiť tak, že celkové penále je najviac k?*
- Partition – *dané celé čísla $a_1, \ldots, a_n$; dajú sa rozdeliť do dvoch skupín s rovnakým súčtom?*
- MaxCut – *obsahuje daný ohodnotený graf rez s váhou aspoň k? (t.j. chceme vrcholy rozdeliť do dvoch skupín a maximalizovať súčet váh hrán, ktoré vedú z jednej skupiny do druhej: $\sum_{\{u,v\} \in E, u \in S, v \notin S} w(\{u,v\})$)*

PSPACE-úplné problémy

Veta 4.6. *Nasledujúce problémy sú* PSPACE*-úplné:*

- $\textsc{TmAcc}_{\textsc{Space}} = \{\langle M \rangle \# x \# 1^s \mid M \text{ je TS, ktorý akceptuje } x \text{ v pamäti } s\}$,
- LbaAcc – *daný je lineárne ohraničený automat M a vstup x; chceme zistiť, či M akceptuje x;*
- CsgGen – *daná je kontextová gramatika G a slovo x; chceme zistiť, či G generuje x;*

■ **Dôkaz.** PSPACE-úplnosť jazyka $\textsc{TmAcc}_{\textsc{Space}}$ je triviálna, podobne ako NP-úplnosť NtmAcc. Zisťovať, či lineárne ohraničený automat akceptuje, je zhruba rovnako ťažké, pretože počas redukcie môžeme „nafúknuť“ vstup: namiesto x budeme mať vstup $x\#^{s-n}$ s dostatočne veľa zarážkami na konci. Hoci výsledok závisí iba od x, formálne má takýto vstup dĺžku s. Ekvivalencia LBA a kontextových gramatík sa dokazuje na Formálnych jazykoch a automatoch.

□

Veta 4.7 (Stockmeyer a Meyer (1973)). *Problém ekvivalencie regulárnych výrazov (tzn. pre dané r_1, r_2, je $L(r_1) \neq L(r_2)$?) je* PSPACE*-úplný.*

■ **Dôkaz.** Redukciou z LBAACC: $L(r_1)$ bude Γ^* a zostrojíme regex r_2, ktorý matchuje *všetko okrem* akceptačného výpočtu $M(x)$ (teda ak M akceptuje x, $L(r_2) \neq \Gamma^* = L(r_1)$ a ak M neakceptuje x, potom r_2 matchuje všetky slová). Zapíšme akceptačný výpočet $C_0 \vdash_M C_1 \vdash_M \vdash_M \cdots \vdash_M C_k$ ako reťazec

$$v = \#C_0\#C_1\#C_2\#\cdots\#C_t\#$$

nad abecedou $\Gamma = Q \cup \Sigma \cup \{\#\}$; každá konfigurácia C_i má dĺžku $n+1$.

Ako vyzerajú slová, ktoré *nie sú* akceptačným výpočtom $M(x)$? Nuž buď

- zle začínajú – slovo nezačína na $\#q_0x\#$; možno je príliš krátke, alebo má na niektorej pozícii zlé písmenko:

  $$s = (\Gamma \mid \varepsilon)^{n+2} \mid (\Gamma - \#)\Gamma^* \mid \#(\Gamma - q_0)\Gamma^* \mid \cdots \mid (\#q_0x_1\cdots x_{i-1}(\Gamma - x_i)\Gamma^*) \mid \cdots$$

- alebo neakceptujú – ľubovoľný reťazec, ktorý neobsahuje akceptačný stav,

  $$f = (\Gamma - q_f)^*$$

- alebo nejaké dve konfigurácie na seba nenadväzujú – reťazec je v tvare

  $$m = \bigcup_{(a,b,c,d,e,f)\in ZK} \Gamma^* abc\Gamma^{n-2}def\Gamma^*,$$

  pre „zlú kombináciu“ písmenok a, b, c, d, e, f; tzn. ak v jednej konfigurácii sú 3 po sebe idúce symboly a, b, c, tak v ďalšej konfigurácii (na rovnakej pozícii o $n+1$ znakov neskôr) nemôžu byť d, e, f; napríklad ak $a, b, c \notin Q$, potom v nasledujúcej konfigurácii by malo byť $b = e$; ak $a \in Q$ a napríklad $\delta(a, b) = (q, b', +1)$, potom v nasledujúcej konfigurácii by malo byť $def = b'qc$.

Výsledný regex je $r_2 = s \mid f \mid m$.

Na druhej strane, problém patrí do PSPACE – regulárne výrazy prevedieme na NKA; chceme nájsť slovo $w \in L(A_1), w \notin L(A_2)$ □

Veta 4.8 (Stockmeyer a Meyer (1973)). *Problém* QBF *je* PSPACE*-úplný.*

■ **Dôkaz.** Ľahko vidieť, že QBF $\in$ PSPACE; majme teda PSPACE stroj M pre jazyk L. Pre každé x zostrojíme ϕ také, že $x \in L \iff \phi \in$ QBF. Vieme, že M použije najviac n^k políčiek pásky a zastaví po 2^{n^k} krokoch. Postupne zostrojíme formule $\phi_i(a, b)$, ktoré sú pravdivé vtedy, keď sa z konfigurácie a vieme dostať do b na $\leq 2^i$ krokov (konfigurácie vieme zapísať pomocou $O(n^k)$ premenných). Potom ak c_0 je počiatočná a c_f jediná konečná konfigurácia (uvažujme napr. M, ktorý na konci vymaže celú pásku a vráti sa na začiatok), tak $x \in L \iff \phi_{n^k}(c_0, c_f) \in$ QBF.

Formulu $\phi_0(a,b)$ máme z Cook-Levinovej vety; $\phi_{i+1}(a,b)$ by sme mohli definovať podobne ako v Savitchovej vete: $\exists c : \phi_i(a,c) \wedge \phi_i(c,b)$, ale bola by exponenciálne veľká. Potrebujeme použiť aj všeobecné kvantifikátory; približne takto: $\phi_{i+1}(a,b) \equiv \exists c \forall x,y : ([x,y] = [a,c] \vee [x,y] = [c,b]) \Rightarrow \phi_i(x,y)$. Ostáva doriešiť technické detaily a presvedčiť sa, že ϕ_{n^k} vieme napísať v dobrom tvare. □

4.3 Relativizácia

Turingov stroj s orákulom je stroj, ku ktorému môžeme pripojiť akúsi čiernu krabičku (orákulum), ktorá rozhoduje jazyk O. Turingov stroj sa môže orákula spýtať otázku: „Patrí slovo q do O?“ a orákulum mu ihneď pravdivo odpovie. Trochu formálnejšie: stroju pridáme špeciálnu pásku a tri špeciálne stavy: query, yes a no; ak na túto pásku napíše slovo q a prejde do stavu query, v nasledujúcom kroku sa ocitne v stave yes alebo no podľa toho, či $q \in O$ alebo $q \notin O$. Podobne definujeme nedeterministické TS s orákulom.

Väčšina dôkazov, ktoré sme zatiaľ videli tzv. „relativizujú“ – sú to nejaké tvrdenia o TS a *presne tak isto* by sa dokázali aj analogické tvrdenia o TS s orákulom. Napríklad veta o časovej/pamäťovej hierarchii hovorí (zhruba), že ak máme viac času/priestoru, potom vieme akceptovať zložitejšie jazyky. Diagonalizáciou vieme zostrojiť jazyk, ktorý patrí do $\mathsf{DTIME}(n^{47})$, ale nepatrí do $\mathsf{DTIME}(n^{46})$. Úplne rovnako by sa zostrojil jazyk, ktorý vieme rozpoznať v čase n^{47} s orákulom O, ale nevieme ho rozpoznať v čase n^{46} s orákulom O.

Z toho, čo sme však dokázali o P^A a NP^A vyplýva, že dôkaz $\mathsf{P} = \mathsf{NP}$ alebo $\mathsf{P} \neq \mathsf{NP}$ *nie je relativizujúci*! Nemôže byť taký, že by sa úplne rovnako dokázalo analogické tvrdenie pre ľubovoľné orákulum.

Toto tvrdenie sa tiež nazýva „relativizačná bariéra“. Naozaj zatiaľ nepoznáme veľa techník, ktoré nerelativizujú a táto bariéra sčasti vysvetľuje, prečo je $\mathsf{P} = \mathsf{NP}$? taký ťažký problém.

Neskôr sa ukázali ďalšie „bariéry“ (tzv. prirodzené dôkazy a algebraická bariéra) – tvrdenia, že $\mathsf{P} \neq \mathsf{NP}$ sa nedá dokázať určitými ďalšími technikami.

Definícia 4.5. *Pre ľubovoľné O definujme P^O a NP^O ako triedu jazykov rozpoznávanú deterministickými, resp. nedeterministickými TS s orákulom O. Ak $\mathcal{C}$ je trieda jazykov, tak $\mathsf{P}^{\mathcal{C}}$ a $\mathsf{NP}^{\mathcal{C}}$ definujeme ako $\bigcup_{O \in \mathcal{C}} \mathsf{P}^O$, resp. $\bigcup_{O \in \mathcal{C}} \mathsf{NP}^O$.*

Jednoduché pozorovania:

- Trieda P^O je uzavretá na komplement.
- Ak $O \in \mathsf{P}$, tak $\mathsf{P}^O = \mathsf{P}$.
- Ak O je úplný pre triedu $\mathcal{C}$, tak $\mathsf{P}^O = \mathsf{P}^{\mathcal{C}}$.
- $\mathsf{P}^{\mathsf{REC}} = \mathsf{NP}^{\mathsf{REC}} = \mathsf{REC}^{\mathsf{REC}} = \mathsf{REC}$
- $\mathsf{P}^{\mathsf{PSPACE}} = \mathsf{NP}^{\mathsf{PSPACE}} = \mathsf{PSPACE}^{\mathsf{PSPACE}} = \mathsf{PSPACE}$

Veta 4.9 (Baker a spol. (1975)). *Existujú jazyky A, B také, že* $\mathsf{P}^A = \mathsf{NP}^A$ *a* $\mathsf{P}^B \neq \mathsf{NP}^B$.

■ **Dôkaz.** Za A stačí zobrať jazyk QBF, ktorý je PSPACE-úplný; platí $\mathsf{P}^{\mathrm{QBF}} = \mathsf{NP}^{\mathrm{QBF}} = \mathsf{PSPACE}$.

Na druhej strane pre jazyk B definujme $U_B = \{1^{|x|} \mid x \in B\}$. Zrejme pre každé B platí $U_B \in \mathsf{NP}^B$ (tipnem x a spýtam sa B). Diagonalizáciou vieme nájsť také B, že $U_B \notin \mathsf{P}^B$. □

Úlohy

- Dokážte, že logspace redukcia je tranzitívna, teda ak $A \leq_{\mathrm{m}}^{\log} B$ a $B \leq_{\mathrm{m}}^{\log} C$, tak $A \leq_{\mathrm{m}}^{\log} C$.
- Uvažujme problém zistiť, či je daná formula splniteľná aspoň dvoma rôznymi ohodnoteniami. Je NP-úplný?
- Nájdite jazyk, ktorý dokázateľne *nie je* NP-úplný.
- Nájdite jazyk, ktorý sa dá redukovať na NP-úplný aj na coNP-úplný problém.
- (Berman a Hartmanis, 1977) Dokážte, že $L \in \mathsf{P/poly}$ práve vtedy, keď L je polynomiálne Turingovsky redukovateľný na nejaký riedky jazyk. Teda $L \in \mathsf{P/poly} \iff \exists$riedky $S : L \leq_{\mathrm{T}}^{\mathrm{P}} S$. Hovoríme, že jazyk je *riedky*, ak je počet slov dĺžky n v jazyku iba polynomiálny.
- Vysvetlite, prečo je nasledujúci argument chybný:

  > SAT vieme vyriešiť v lineárnej pamäťi. SAT je NP-úplný problém a preto $\mathsf{NP} \subseteq \mathsf{DSPACE}(n)$. Podľa pamäťovej hierarchie je $\mathsf{DSPACE}(n) \subset \mathsf{PSPACE}$ a teda $\mathsf{NP} \neq \mathsf{PSPACE}$.

  (Pripomeňme, že otázka, či $\mathsf{P} \neq \mathsf{PSPACE}$ je stále otvorený problém.)
- Sú P a NP uzavreté na homomorfizmus?
- Dokážte, že nasledujúce problémy sú NL-úplné:
  1. Rozhodnúť, či daný nedeterministický automat A akceptuje dané slovo w.
  2. Rozhodnúť, či je daný orientovaný graf silne súvislý.
  3. Rozhodnúť, či daný orientovaný graf obsahuje cyklus.
- Uvažujme obmenu problému vrcholového pokrytia s tým, že na vstupe je graf G, číslo k a dostaneme prísľub, že buď má G vrcholové pokrytie najviac k vrcholmi alebo má každé pokrytie minimálne $3k$ vrcholov. Je tento problém NP-úplný alebo v P?

Literatúra

Baker, Theodore, John Gill, a Robert Solovay. 1975. "Relativizations of the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ question." *SIAM Journal on computing* 4(4), s. 431–442.

Beigel, Richard a David Eppstein. 2005. "3-coloring in time $O(1.3289^n)$." *Journal of Algorithms* 54(2), s. 168–204.

Berman, Leonard a Juris Hartmanis. 1977. "On isomorphisms and density of NP and other complete sets." *SIAM Journal on Computing* 6(2), s. 305–322.

Cook, Stephen A. 1971. "The complexity of theorem-proving procedures." In *Proceedings of the third annual ACM symposium on Theory of computing*. ACM, s. 151–158.

Harvey, David a Joris van der Hoeven. 2019. "Integer multiplication in time $O(n \log n)$." .

Karp, Richard M. 1972. "Reducibility among combinatorial problems." In *Complexity of computer computations*. Springer, s. 85–103.

Levin, Leonid Anatolevich. 1973. "Universal sequential search problems." *Problemy peredachi informatsii* 9(3), s. 115–116.

Stockmeyer, Larry J a Albert R Meyer. 1973. "Word problems requiring exponential time (preliminary report)." In *Proceedings of the fifth annual ACM symposium on Theory of computing*. ACM, s. 1–9.

Časť II

Alternácia
(Svet medzi P a PSPACE)

Úvod

V úvode sme si vraveli, že cieľom teórie zložitosti je (v ideálnom prípade) zistiť, aké ťažké sú rôzne problémy. Avšak keďže nám dokazovanie dolných odhadov zatiaľ veľmi nejde, snažíme sa aspoň roztriediť problémy do rôznych tried ekvivalencie.

Poďme si to skúsiť. Uvažujme nasledovné problémy:

- MIN-DNF – daná je DNF formula ϕ a číslo k; existuje ekvivalentná DNF formula veľkosti $\leq k$?
- 1LTA-GRAMMAR-INEQUIVALENCE – dané sú bezkontextové gramatiky G_1 a G_2 nad 1-písmenovou abecedou; je $L(G_1) \neq L(G_2)$?
- 3-COLORING-EXTENSION – daný je graph G; dá sa každé 3-ofarbenie listov rozšíriť na 3-ofarbenie celého grafu?
- VC-DIMENSION[3] – daná je kolekcia $\mathcal{C} = \{S_1, S_2, \ldots\}$ podmnožín konečnej množiny U (S_i sú reprezentované úsporne booleovskými obvodmi); je $VC(\mathcal{C}) \geq k$? T.j. existuje $X \subseteq U$, $|X| \geq k$, $\forall S \subseteq X : \exists i : S = S_i \cap X$?

Aká je zložitosť týchto problémov?

Na jednej strane ľahko vidno, že všetky tieto problémy sa dajú riešiť v exponenciálnom čase, dokonca v polynomiálnom priestore, teda patria do PSPACE. Na druhej strane, všetky vyzerajú aspoň NP-ťažké. No dobre. Ale sú teda NP-úplné alebo PSPACE-úplné? Alebo že by „niečo medzi“?

Vezmime si prvý problém, MIN-DNF. Ak by sme ho chceli riešiť pomocou NTS, mohli by sme si tipnúť formulu ψ, ale ako overíme, či je ekvivalentná s ϕ? Na to by sme potrebovali zistiť, či $\phi \equiv \psi$ je tautológia – čo je coNP-úplný problém.

[3] Vapnik-Červonenkisova dimenzia sa používa v teórii učenia i výpočtovej geometrii. Meria „výrazovú silu“, „flexibilitu“ súboru funkcií. Dôsledné pochopenie tohto pojmu nie je nutné pre ďalšiu diskusiu (ide len o príklad prakticky motivovaného problému), avšak ak by to niekoho zaujímalo, tu je definícia ešte raz a trochu ľudskejšie: Majme množinu X a ofarbime jej body dvomi farbami (na červeno alebo na modro). Budeme hovoriť, že S_i vie rozlíšiť modré body od červených, ak $S_i \cap X$ je presne množina modrých bodov. Hovoríme, že kolekcia $\mathcal{C}$ *rozbije* množinu X, ak pre ľubovoľné zafarbenie X existuje $S_i \in \mathcal{C}$, ktorá odlíši modré body od červených. VC-dimenzia je veľkosť najväčšej množiny, ktorú dokáže $\mathcal{C}$ rozbiť.

Čo 3-Coloring-Extension? Už len zistiť, či sa jedno konkrétne 3-farbenie dá rozšíriť na 3-farbenie celého grafu je NP-úplný problém. My však potrebujeme overiť *všetky* ofarbenia listov. Intuitícia vraví, že toto NTS v polynomiálnom čase nezvládne.

Vyzerá to, že uvedené problémy sú ťažšie ako NP. Sú však PSPACE-úplné? Pravdupovediac, ani to nevyzerá byť pravda. Môžeme ich totiž vyjadriť takto:

- Min-Dnf:
$$\exists \text{ formula } \psi$$
$$\forall \text{ ohodnotenie } x : \phi(x) = \psi(x)$$
- 1Lta-Grammar-Inequivalence:
$$\exists \text{ slovo } w \text{ a odvodenie v } G_1 \text{ alebo } G_2$$
$$\forall \text{ odvodenie v tej druhej gramatike nedostaneme } w$$
- 3-Coloring-Extension:
$$\forall \text{ ofarbenie listov}$$
$$\exists \text{ ofarbenie grafu}$$
- VC-Dimension:
$$\exists X\ \forall S\ \exists i \text{ také, že dačo platí}$$

Dá sa dokázať, že prvé dva problémy sú ekvivalentné (cez polynomiálnu redukciu) tzv. problému $\exists\forall$Sat: pre danú booleovskú formulu $\phi(x, y)$ zistiť, či existuje ohodnotenie $x_1, \ldots, x_n$ také, že pre všetky ohodnotenia $y_1, \ldots, y_m$ je formula $\phi(x, y)$ splnená. Inými slovami, či je pravda, že $\exists x \forall y : \phi(x, y)$. 3-Coloring-Extension je ekvivalentný problému $\forall\exists$Sat, t.j. zistiť, či je daná formula $\forall x \exists y : \phi(x, y)$ pravdivá. VC-Dimension je zase ekvivalentný problému $\exists\forall\exists$Sat.

Naproti tomu NP-úplný Sat či coNP-úplná tautológia majú iba jeden existenčný či univerzálny kvantifikátor. A opačný extrém: v PSPACE-úplnom QBF sa môže striedať ľubovoľne veľa kvantifikátorov.

Pripomeňme, že v súčasnosti nevieme dokázať ani či $\mathsf{P} \neq \mathsf{PSPACE}$. Teoreticky je teda stále možné, že všetky spomínané problémy sú rovnako ťažké. Avšak tak ako prevažuje domnienka, že $\mathsf{P} \neq \mathsf{NP}$ a $\mathsf{NP} \neq \mathsf{PSPACE}$, veríme, že $\exists\forall$Sat ani $\exists\forall\exists$Sat nepatria do NP. Tieto problémy sú úplné pre triedy, ktoré voláme $\mathbf{\Sigma}_2^{\mathsf{P}}$ a $\mathbf{\Sigma}_3^{\mathsf{P}}$, pričom $\mathsf{NP} \cup \mathsf{coNP} \subseteq \mathbf{\Sigma}_2^{\mathsf{P}} \subseteq \mathbf{\Sigma}_3^{\mathsf{P}} \subseteq \mathsf{PSPACE}$ a veríme, že všetky inklúzie sú ostré.

V tejto časti budeme študovať, ako vyzerá svet medzi P a PSPACE. Dôležitý koncept, ktorý nám pomôže pri štúdiu takýchto otázok je *alternácia*. Ide o veľmi prirodzené zovšeobecnenie nedenterminizmu. Triedu NP sme definovali cez NTS, ktoré si „tipnú“ správnu odpoveď a v polynomiálnom čase ju overia. Mohli by sme si to predstaviť tiež tak, že pri nedeterministickom rozhodovaní sa program rozvetví a paralelne vyskúša všetky možnosti. Akceptuje, ak *aspoň jedna* vetva akceptuje.

Problémy z coNP sme definovali jednoducho ako komplementy jazykov z NP. Bolo by však oveľa krajšie (a užitočnejšie), keby sme mali nejaký výpočtový model, ktorý coNP charakterizuje. Akýsi „ko-nedeterministický“ Turingov stroj. A čosi také vskutku existuje. Mohli by sme si predstaviť niečo ako NTS – keď má program viac možností pokračovania, rozvetví sa a paralelne vyskúša všetky možnosti – s tým rozdielom, že akceptujeme práve vtedy, keď *všetky* vetvy akceptujú. Napríklad coNP-úplný problém tautológie vieme vyriešiť pomocou takéhoto „ko-NTS“ v polynomiálnom čase: stačí paralelne vyskúšať všetky ohodnotenia a skontrolovať, že pri každom je formula splnená.

Odtiaľto sme už len krôčik k definovaniu alternujúceho Turingovho stroja (ATS). Ten dokáže striedať medzi tzv. existenčnými a univerzálnymi stavmi, pričom v existenčných stavoch paralelne skúša všetky možnosti a akceptuje, ak niektorá (aspoň jedna) vetva akceptuje, v univerzálnych stavoch paralelne skúša všetky možnosti a akceptuje, ak všetky vetvy akceptujú. Presná definícia bude v nasledujúcej kapitole. Tu len poznamenajme, že NTS sú ATS, ktoré sú počas celého výpočtu iba v existenčných stavoch. Naopak „ko-NTS“ sú ATS, ktoré sú celý čas v univerzálnych stavoch. Problém ako $\exists\forall$SAT vieme vyriešiť ATS, ktorý v existenčných stavoch natipuje ohodnotenie x, potom sa prepne (alternuje) do univerzálneho stavu a paralelne vyskúša všetky ohodnotenia y. QBF zase vieme riešiť v polynomiálnom čase, ak môžeme neobmedzene alternovať.

Na záver uveďme ešte jednu oblasť, kde je takéto rozmýšľanie veľmi prirodzené, a to sú hry (ku ktorým sa dostaneme v štvrtej časti). Konkrétne hry, kde hrajú dvaja hráči proti sebe. Vezmime si napr. taký šach. Predstavme si, že dostaneme pozíciu v šachu a otázku, či vie dať biely mat na tri ťahy. V preklade to znamená, či

existuje taký ťah bieleho, že pre každú odpoveď čierneho,

existuje taký ťah bieleho, že nech spraví čierny čokoľvek,

existuje ťah bieleho, ktorý matuje.

Skrátene, či

$$\exists\text{ťah B}\ \forall\text{ťah Č}\ \exists\text{ťah B}\ \forall\text{ťah Č}\ \exists\text{ťah B} : \text{mat}.$$

Alebo dostaneme pozíciu a otázku, či má biely vyhrávajúcu stratégiu – či si dokáže vynútiť výhru – na ľubovoľne veľa ťahov.

Jedna možnosť ako rozmýšľať o takýchto problémoch je cez rôzne prehľadávania grafu pozícií; druhá možnosť je rozmýšľať v „reči“ alternujúcich TS: existenčné stavy zodpovedajú ťahom bieleho, univerzálne stavy ťahom čierneho.

Kapitola 5

Alternujúce Turingove stroje

Alternujúce Turingove stroje definujeme rovnako ako tie nedeterministické s tým, že stavy rozdelíme na existenčné ($\exists$) a univerzálne ($\forall$); formálne pridáme označenie stavov

$$\text{typ} : Q \to \{\exists, \forall\}.$$

Podobne ako nedeterministické stroje, aj tie alternujúce môžu mať veľa rôznych výpočtov na jednom vstupe.

To, či stroj akceptuje, zistíme tak, že sa pozrieme na celý strom všetkých výpočtov a začneme konfigurácie postupne od konca od listov ofarbovať – či sú akceptačné alebo odmietavé. Konfigurácie, keď stroj zastane, sú jasné – skončí buď v akceptačnom alebo odmietavom stave (pozri obrázok 5.1). Ostatné konfigurácie označíme postupne tak, že

- existenčná konfigurácia je akceptačná, ak existuje aspoň jedna nasledujúca konfigurácia, ktorú sme už označili ako akceptačnú,
- univerzálna konfigurácia je akceptačná, ak všetky možné kroky vedú do akceptačnej konfigurácie.

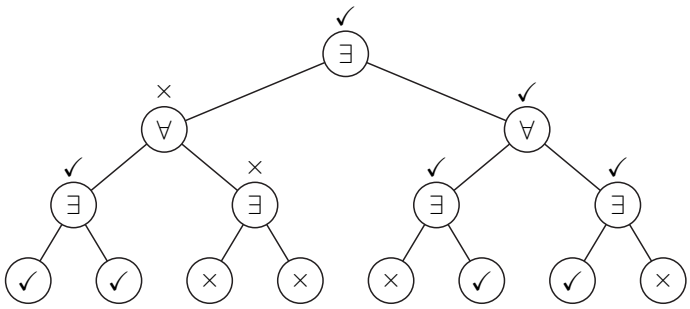
Stroj akceptuje, ak je jeho počiatočná konfigurácia akceptačná.[1]

[1] [Iba pre skalných formalistov.] Ak by sme chceli byť veľmi formálni, môžeme definovať množinu konfigurácií $\mathcal{C}$ stroja M a ich ofarbenia $\ell : \mathcal{C} \to \{\bot, 0, 1\}$, kde 0/1 znamená odmietavá/akceptačná konfigurácia a $\bot$ znamená „zatiaľ neviem“. Definujeme usporiadanie $\bot \sqsubseteq 0$ a $\bot \sqsubseteq 1$ a $\ell \sqsubseteq \ell'$, ak sa ℓ a ℓ' na ofarbených konfiguráciach zhodujú, ale ℓ' má možno ofarbené ešte niečo navyše: $\forall \alpha \in \mathcal{C} : \ell(\alpha) \sqsubseteq \ell'(\alpha)$. Definujme zobrazenie na ofarbeniach τ:

$$\tau(\ell)(\alpha) = \begin{cases} \bigwedge_{\alpha \vdash_M \beta} \ell(\beta) & \text{ak } \alpha \text{ je } \forall\text{-konfigurácia} \\ \bigvee_{\alpha \vdash_M \beta} \ell(\beta) & \text{ak } \alpha \text{ je } \exists\text{-konfigurácia} \end{cases}$$

No a potom už iba vezmeme pevný bod zobrazenia τ. Dostaneme ofarbenie ℓ^*, ktoré môžeme získať tak, že začneme z prázdneho označenia ℓ_0 ($\bot$ pre všetky konfigurácie) a opakovane aplikujeme τ:

$$\ell_0 \sqsubseteq \tau(\ell_0) \sqsubseteq \tau(\tau(\ell_0)) \sqsubseteq \tau(\tau(\tau(\ell_0))) \sqsubseteq \cdots$$



Obr. 5.1: Výpočtový strom alternujúceho TS. Výpočet začína v koreni, môžeme si predstaviť, že vždy, keď má stroj na výber viacero možných pokračovaní, spustí nové podprocesy, ktoré pokračujú paralelne a počká kým skončia. Každý vrchol je jeden proces, ktorý čaká, kým skončí výpočet pod ním. Listy tohto stromu sú koncové konfigurácie: akceptačné (✓) a odmietajúce (×). Každý proces, ktorý skončí, odovzdá materskému procesu informáciu, či akceptuje alebo odmieta. Keď skončia všetky deti nejakého vrcholu, rozhodne sa na základe odpovedí (a podľa toho, či je existenčný, alebo univerzálny) aj ich materský proces a skončí. Výsledok výpočtu sa dozvieme z koreňa.

Jeden spôsob, ako sa pozerať na alternujúce stroje, je ako na paralelný model: v každom stave s viacerými možnosťami sa program „fork-ne“ a vytvorí nové podprocesy, ktoré paralelne skontrolujú všetky možnosti; keď tieto podprocesy skončia, ich rodič jednoducho vyhodnotí, či akceptuje (podľa typu stavu). Po k krokoch môžeme mať až exponenciálne veľa paralelne bežiacich procesov. Ukážeme, že alternujúce Turingove stroje skutočne sú „rozumný“ paralelný model v zmysle tézy o paralelných výpočtoch.

5.1 Alternujúci čas a priestor

Definícia 5.1 (ATIME, ASPACE). *Nech $f : \mathbb{N} \to \mathbb{N}$ je funkcia. Hovoríme, že alternujúci stroj pracuje v čase/pamäti $f(n)$, ak každý výpočet na každom slove x zaberie čas/pamäť $f(|x|)$. Definujeme* $\mathsf{ATIME}(f(n))$ *a* $\mathsf{ASPACE}(f(n))$ *ako triedy jazykov rozhodnuteľných na alternujúcom Turingovom stroji v čase, respekíve priestore $O(f(n))$. Špeciálne*

- $\mathsf{AL} = \mathsf{ASPACE}(\log n)$ – *alternujúci logaritmický priestor,*
- $\mathsf{AP} = \bigcup_k \mathsf{ATIME}(n^k)$ – *alternujúci polynomiálny čas,*
- $\mathsf{APSPACE} = \bigcup_k \mathsf{ASPACE}(n^k)$ – *alternujúci polynomiálny priestor,*
- $\mathsf{AEXP} = \bigcup_k \mathsf{ATIME}(2^{n^k})$ – *alternujúci exponenciálny čas.*

Potom ℓ^* je suprémum (najmenšie horné ohraničenie) tejto reťaze.

Automatická prvá otázka pri takomto novom modeli: aká je jeho sila? Čo dokáže alternujúci Turingov stroj v polynomiálnom čase? Zrejme aspoň toľko, čo nedeterministický (čo je špeciálny prípad), ale zdá sa, že oveľa viac. Zatiaľčo nedeterministický stroj dokáže v polynomiálnom čase zistiť, či

$$\exists x_1 \exists x_2 \cdots \exists x_k : \psi(x_1, \ldots, x_k),$$

alternujúci stroj dokáže zistiť, či

$$\exists x_1 \forall x_2 \cdots \exists x_k : \psi(x_1, \ldots, x_k),$$

teda riešiť problém QBF v polynomiálnom čase!

Algoritmus je jednoduchý: Pre $(\exists x_i)$ si vyberieme správnu hodnotu v existenčnom stave (nedeterministicky); teda stroj bude mať dve možné pokračovania, $x_i = 0$ a $x_i = 1$, a akceptujeme, ak aspoň jedna voľba x_i vyhovuje. Naopak pre $(\forall x_i)$ sa preklopíme do univerzálneho stavu a vyskúšame obe možnosti $x_i = 0$ a $x_i = 1$ naraz; teda stroj bude mať dve možné pokračovania a akceptujeme, ak obe voľby x_i vyhovujú. Nakoniec len v polynomiálnom čase vyhodnotíme formulu ψ pri danom ohodnotení premenných a akceptujeme, ak je pravdivá. Každý jeden výpočet je teda len polynomiálne dlhý, ale všetky rôzne výpočty zodpovedajú všetkým možným ohodnoteniam.

Vidíme teda, že QBF $\in$ AP a keďže tento problém je PSPACE-úplný, tak PSPACE $\subseteq$ AP. Na druhej strane, v polynomiálnom priestore zjavne vieme prejsť *všetky* polynomiálne dlhé výpočty a prehľadávaním do hĺbky ofarbiť všetky konfigurácie ako akceptačné/odmietavé (pamäť je úmerná dĺžke cesty od koreňa k listu, skúste si rozmyslieť detaily.) Takže AP = PSPACE.

Iný pohľad na to isté je, že ak vieme výpočty popísať booleovskými formulami ako v Cook-Levinovej vete, potom (intuitívne) striedanie existenčných a univerzálnych kvantifikátorov môžeme využiť na zachytenie striedania existenčných a univerzálnych stavov. (Skúste si rozmyslieť detaily.)

Toto tvrdenie sa dá zovšeobecniť a súvis medzi alternujúcim časom a deterministickým priestorom spresniť. Dokážeme ho priamo simuláciou.

Veta 5.1 (Chandra a spol. (1981)). *Pre $t(n) \geq n$ a $s(n) \geq \log n$ platí:*

1. $\mathsf{ATIME}(t(n)) \subseteq \mathsf{DSPACE}(t(n))$ *a*
2. $\mathsf{DSPACE}(s(n)) \subseteq \mathsf{ATIME}(s(n)^2)$.

Z toho vyplýva, že alternujúci čas je zhruba rovnaký ako deterministický priestor (až na polynomiálne faktory):

$$\mathsf{ATIME}(t(n)^{O(1)}) = \mathsf{DSPACE}(t(n)^{O(1)}).$$

Špeciálne AP = PSPACE *a* AEXP = EXPSPACE.

■ **Dôkaz.** 1. Dôkaz je zovšeobecnením dôkazu tvrdenia $\mathsf{NTIME}(t(n)) \subseteq \mathsf{DSPACE}(t(n))$. Budeme sekvenčne prechádzať všetky výpočty alternujúceho stroja. Na páske si

stačí pamätať voľby, ktoré sme už v existenčných a univerzálnych stavoch urobili a označenie, či je konfigurácia akceptačná/odmietavá/zatiaľ nevieme. Ak si predstavíme celý strom výpočtov alternujúceho stroja, my si budeme prechádzať cesty od koreňa postupne ku všetkým listom a pamäť navyše bude úmerná dĺžke cesty, teda $O(t(n))$.

2. Dôkaz je rovnaký ako v Savitchovej vete, akurát použijeme paralelizmus. Chceme vedieť, či sa dá dostať z počiatočnej do akceptačnej konfigurácie, všeobecne, či $C_1 \vdash_M^k C_2$. Pomocou existenčných stavov natipujeme prostrednú konfiguráciu C a pomocou univerzálnych stavov paralelne skontrolujeme, či $C_1 \vdash_M^{\lfloor k/2 \rfloor} C$ a zároveň $C \vdash_M^{\lceil k/2 \rceil} C_2$.

Skúste si rozmyslieť, že toto je rovnaký trik, aký sme použili pri dôkaze, že problém QBF je PSPACE-úplný. □

Pripomeňme si, že podľa tézy o paralelných výpočtoch sú čas na „rozumnom“ paralelnom modeli a priestor na „rozumnom“ sekvenčnom modeli zhruba ekvivalentné. V tomto zmysle sú alternujúce stroje „rozumný“ model paralelných výpočtov.

Veta 5.2 (Chandra a spol. (1981)). *Pre $t(n) \geq n$ a $s(n) \geq \log n$ platí:*

1. $\mathsf{ASPACE}(s(n)) \subseteq \mathsf{DTIME}(2^{O(s(n))})$,
2. $\mathsf{DTIME}(t(n)) \subseteq \mathsf{ASPACE}(\log t(n))$.

Teda alternujúci priestor je zhruba rovnaký ako exponenciálny deterministický čas:

$$\mathsf{ASPACE}(s(n)^{O(1)}) = \mathsf{DTIME}(2^{O(s(n))}).$$

Špeciálne $\mathsf{AL} = \mathsf{P}$ *a* $\mathsf{APSPACE} = \mathsf{EXP}$.

■ **Dôkaz.** 1. Stroj s pamäťou $s(n)$ môže dosiahnuť najviac $C = 2^{O(s(n))}$ konfigurácií. Všetky si ich vygenerujeme a následne budeme ofarbovať tie akceptačné: začneme s konfiguráciami, kde stroj zastane a akceptuje a postupne ofarbujeme ďalšie a ďalšie podľa definície ATS. Skončíme, keď ofarbíme začiatočnú konfiguráciu (vtedy vieme, že výpočet bol akceptačný), alebo keď už nevieme ofarbiť žiadnu novú konfiguráciu (vtedy sme už ofarbili všetky akceptačné konfigurácie[2] a teda vieme, že výpočet nebol akceptačný). Čas bude polynomiálny od C, čo je stále $2^{O(s(n))}$.

2. Predstavme si t krokov výpočtu ako tabuľku $t \times t$, kde i-ty riadok je konfigurácia v čase i. Ako sme si ukázali v kapitole 4.2, tento výpočet sa dá popísať obvodom alebo booleovskou formulou, pričom obsah „políčka“ j v čase i závisí len od obsahu konštantne veľa políčok okolo j v čase $i-1$ a daný obvod bol veľmi uniformný. Ekvivalentný ATS bude tento obvod vyhodnocovať, pričom pôjde od posledného riadku, poslednej konfigurácie k počiatočnej a pre každé políčko bude kontrolovať jeho správnosť „paralelne“. Uvedomme si, že s logaritmickou pamäťou máme dosť miesta na krok výpočtu a index

[2] dosiahli sme pevný bod ℓ^* zobrazenia τ, viď poznámku pod čiarou na strane 1

políčka, ale nemáme dosť miesta, aby sme si mohli vypísať jednu celú konfiguráciu, alebo celú históriu jedného políčka. Využijeme preto existenčné stavy, v ktorých budeme tipovať obsah jednotlivých políčok, vždy skontrolujeme, či lokálne výpočet sedí a pomocou univerzálnych stavov rekurzívne skontrolujeme, že všetky políčka v predošlom kroku výpočtu sme natipovali správne.

□

Zhrnutie

Zatiaľ sa nám podarilo dokázať, že základné alternujúce triedy (AL, AP, APSPACE, AEXP) sa zhodujú s tými klasickými – len trochu posunuto:

$$\begin{array}{ccccccccccc} \mathsf{L} & \subseteq & \mathsf{P} & \subseteq & \mathsf{PSPACE} & \subseteq & \mathsf{EXP} & \subseteq & \mathsf{EXPSPACE} & \subseteq \cdots \\ & & \| & & \| & & \| & & \| & \\ & & \mathsf{AL} & \subseteq & \mathsf{AP} & \subseteq & \mathsf{APSPACE} & \subseteq & \mathsf{AEXP} & \subseteq \cdots \end{array}$$

To je dobrá správa: dostali sme nové charakterizácie, nový pohľad na „staré“ triedy. Dáva nám to nový spôsob, ako rozmýšľať o týchto triedach, čo môžeme využiť v ďalších dôkazoch.

V hornom riadku máme deterministické výpočty (bez alternácie), v dolnom riadku máme neobmedzenú alternáciu. V nasledujúcej kapitole sa pozrieme na to, čo sa stane, ak alternáciu síce povolíme, ale len v obmedzenej miere.

Úlohy

- Dokážte, že pre ľubovoľné k trieda PSPACE obsahuje problémy, ktoré sa *nedajú* riešiť (neuniformnými) obvodmi veľkosti n^k. Inými slovami, pre každé k je rozdiel $\mathsf{PSPACE} - \mathsf{SIZE}(n^k)$ neprázdny. Pripomeňme, že $\mathsf{SIZE}(n)$ obsahuje aj nerozhodnuteľné jazyky, avšak $\mathsf{PSPACE} \not\subset \mathsf{SIZE}(n^k)$ pre žiadne fixné k. (Otázka, či $\mathsf{PSPACE} \subset \mathsf{P/poly}$ je otvorená, keďže nedokážeme vylúčiť ani, že $\mathsf{P} = \mathsf{PSPACE}$),

- Dokážte, že alternujúce Turingove stroje, ktoré robia $A(n)$ alternácií a používajú $S(n)$ pamäte (páskovo konštruovateľné funkcie, $S(n) \geq \log n$) sa dajú simulovať deterministicky v pamäti $O(A(n)S(n) + S(n)^2)$. Všimnite si, že pre $A(n) = 1$ dostaneme Savitchovu vetu $\mathsf{NSPACE}(S(n)) \subseteq \mathsf{DSPACE}(S(n)^2)$. Napríklad pre polynomiálny priestor sme sa učili, že $\mathsf{PSPACE} = \mathsf{NPSPACE}$. Podľa tejto vety aj keby sme pridali polynomiálne veľa alternácie, stále ostávame v PSPACE.

- Dokážte, že pre každý alternujúci Turingov stroj pracujúci v čase $t(n)$ a pamäti $s(n)$ existuje ekvivalentný stroj, ktorý sa počas každého výpočtu maximálne raz, na konci pozrie na jeden znak vstupu a pracuje v čase $O(t(n))$ a pamäti $O(s(n))$. (Môžeme si predstaviť model, kde čítacia hlava nevidí vstup a načíta písmeno, až keď stroj prejde do špeciálneho stavu.)

Literatúra

Chandra, Ashok K, Dexter C Kozen, a Larry J Stockmeyer. 1981. "Alternation." *Journal of the ACM (JACM)* 28(1), s. 114–133.

Kapitola 6

Polynomiálna hierarchia

V tejto a nasledujúcej kapitole budeme študovať tzv. polynomiálnu hierarchiu – veľmi prirodzenú a užitočnú hierarchiu väčších a väčších tried medzi P a PSPACE.

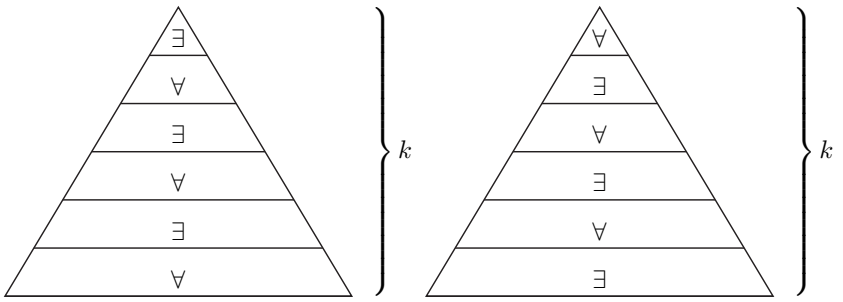
To, že tieto triedy sú naozaj prirodzené, uvidíme z množstva ekvivalentných definícií, ktoré ich vystihujú. Na „prirodzené“ objekty v matematike a informatike môžeme často nahliadať viacerými spôsobmi. Napríklad na kombinačné čísla sa môžeme pozerať cez Pascalov trojuholník, cez binomickú vetu, či cez kombinatoriku (počet kombinácií). Na sústavu lineárnych rovníc sa môžeme pozerať cez matice, cez riadky, cez stĺpce, cez lineárne zobrazenia, cez determinanty,.... Na regulárne jazyky sa môžeme pozerať cez DKA, NKA, regulárne gramatiky, či regulárne výrazy. Rekurzívne funkcie môžeme zadefinovať cez milión možných ekvivalentných modelov. No a podobne je na tom aj polynomiálna hierarchia, ukážeme si sedem rôznych spôsobov, ako sa na ňu pozerať. V tomto zmysle je táto hierarchia prirodzená.

Polynomiálna hierarchia je navyše vznikla priamočiarou analógiou z tzv. aritmetickej hierarchie, ktorá sa študuje v teórii vypočítateľnosti. Jediný rozdiel je, že namiesto „vypočítateľné v konečnom čase“ sa budeme venovať polynomiálnemu času.

6.1 Alternácia

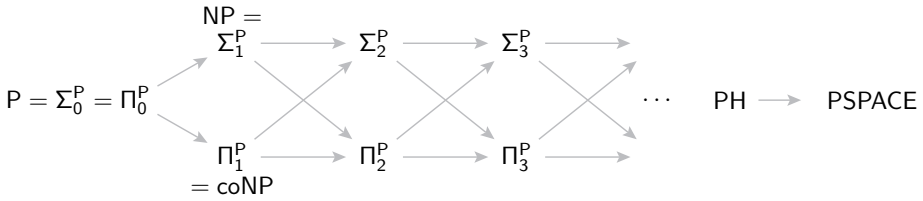
Definícia 6.1 (Polynomiálna hierarchia I.). *Hovoríme, že alternujúci Turingov stroj je Σ_k-stroj (resp. Π_k-stroj), ak začne v existenčnom (resp. univerzálnom) stave a počas každého výpočtu najviac $(k-1)$-krát prejde z existenčného stavu do univerzálneho alebo naopak. Teda každý výpočet sa dá rozdeliť na k úsekov, v ktorých je stroj striedavo v $\exists$- a $\forall$-stavoch; hovoríme, že k-krát alternuje (pozri obrázok 6.1).*

Σ_k^{P} a Π_k^{P} potom môžeme definovať ako triedy jazykov akceptované Σ_k-, resp. Π_k-strojom v polynomiálnom čase. Trieda PH (polynomiálna hierarchia) obsa-



(a) Σ_k-stroj začína v $\exists$-stave.. (b) Π_k-stroj začína v $\forall$-stave.

Obr. 6.1: Stroje s obmedzeným alternovaním. Σ_k- a Π_k-stroj sú také alternujúce TS, že každý ich výpočet sa dá rozdeliť na najviac k $\exists/\forall$ úsekov, kde sa typ stavu nemení. Hovoríme, že iba k-krát alternujú.



Obr. 6.2: Polynomiálna hierarchia

huje všetky jazyky, ktoré sú z Σ_k^P *pre nejaké* i*, t.j.*

$$PH = \bigcup_{k=0}^{\infty} \Sigma_k^P.$$

Špeciálne Σ_1-stroje sú celý čas v existenčnom stave – sú to teda nedeterministické stroje a $\Sigma_1^P = NP$. Zrejme čím viac alternácií povolíme, tým dostaneme potenciálne väčšiu triedu. Rozmyslite si, že Σ_k^P aj Π_k^P sú podmnožinou aj Σ_{k+1}^P aj Π_{k+1}^P:

$$\Sigma_k^P \cup \Pi_k^P \subseteq \Sigma_{k+1}^P \cap \Pi_{k+1}^P.$$

Preto nezáleží na tom, či PH definujeme cez Σ_k^P alebo Π_k^P:

$$PH = \bigcup_{k=0}^{\infty} \Sigma_k^P = \bigcup_{k=0}^{\infty} \Pi_k^P.$$

Tieto vzťahy sú zobrazené na obrázku 6.2 (triedy Δ_k^P definujeme neskôr). Na spodku máme P, zjednotenie celej hierarchie je PH a nad tým je PSPACE.

Poznamenajme, že $L \in PH$, ak existuje k (ľubovoľne veľké, ale fixné) také, že problém L sa dá riešiť v polynomiálnom čase s k alternáciami. Naproti tomu v PSPACE je počet alternácií neobmedzený a môže rásť s veľkosťou vstupu (napríklad pri QBF je počet alternácií rovný počtu kvantifikátorov).

6.2 Certifikáty

Druhá možnosť, ako definovať polynomiálnu hierarchiu je cez certifikáty. Trieda NP sa dá zadefinovať takto: $L \in \mathsf{NP}$ práve vtedy, keď existuje polynomiálne dlhý „dôkaz“, ktorý vieme overiť v polynomiálnom čase; inými slovami, existuje polynomiálna relácia R taká, že $x \in L \iff \exists y : (x, y) \in R$. (Tento dôkaz môže obsahovať nedeterministické rozhodnutia Turingovho stroja; na druhej strane nedeterministický Turingov stroj vie tento dôkaz uhádnuť a overiť.)

Keďže coNP obsahuje doplnky jazykov z NP a $\neg\exists x : \phi(x)$ je ekvivalentné $\forall x : \neg\phi(x)$, trieda coNP obsahuje jazyky L, pre ktoré existuje polynomiálna relácia R taká, že $x \in L \iff \forall y : (x, y) \in R$. Je potom prirodzené triedy NP a coNP zovšeobecniť:

Definícia 6.2 (Polynomiálna hierarchia II.). *Hovoríme, že R je polynomiálna relácia, ak je polynomiálne ohraničená, t.j. existuje polynóm p taký že pre každé $(x, y_1, \ldots, y_n) \in R$ je $|y_1| + \cdots + |y_n| \leq p(|x|)$ a navyše je príslušnosť do R rozhodnuteľná v čase polynomiálnom od $|x|$.*

Triedy Σ_k^P a Π_k^P definujeme nasledovne:

- Σ_k^P *je trieda jazykov, pre ktoré existuje polynomiálna relácia R taká, že* $x \in L \iff \exists y_1 \forall y_2 \exists \cdots Q y_k(x, y_1, \ldots, y_k) \in R$
- Π_k^P *je trieda jazykov, pre ktoré existuje polynomiálna relácia R taká, že* $x \in L \iff \forall y_1 \exists y_2 \forall \cdots Q y_k(x, y_1, \ldots, y_k) \in R$

Špeciálne definujeme $\Sigma_0^P = \Pi_0^P = \mathsf{P}$.

Z tejto definície je zjavné, že $L \in \Sigma_k^P \iff \overline{L} \in \Pi_k^P$, teda

$$\Sigma_k^P = \mathsf{co}\Pi_k^P \qquad \text{a} \qquad \Pi_k^P = \mathsf{co}\Sigma_k^P.$$

Namiesto $(k+1)$-árnej relácie R môžeme tiež použiť indukciu. Môžeme pritom použiť operátory $\exists$ a $\forall$ definované sekcii 2.2:

Definícia 6.3 (Polynomiálna hierarchia III.). *Definujme*

- $\Sigma_0^P = \Pi_0^P = \mathsf{P}$,
- $\Sigma_{k+1}^P = \exists \cdot \Pi_k^P \qquad a \qquad \Pi_{k+1}^P = \mathsf{co} \cdot \Sigma_{k+1}^P = \forall \cdot \Sigma_k^P$.

Ak reláciu R v definícii II. zapíšeme pomocou booleovskej formuly (ako v dôkaze Cook-Levinovej vety), dostaneme, že $\exists\forall$SAT je Σ_2^P-úplný, $\forall\exists$SAT je Π_2^P-úplný, $\exists\forall\exists$SAT je Σ_3^P-úplný, atď. Vo všeobecnosti definujeme Σ_kSAT ako problém zistiť, či je daná formula

$$\exists \vec{x}_1 \forall \vec{x}_2 \cdots Q \vec{x}_k : \phi(\vec{x}_1, \ldots, \vec{x}_k)$$

s k striedajúcimi kvantifikátormi pravdivá a podobne definujeme Π_kSAT s tým, že formula začína univerzálnym kvantifikátorom. Problém Σ_kSAT je Σ_k^P-úplný a Π_kSAT je Π_k^P-úplný.

Polynomiálnu hierarchiu vieme ekvivalentne definovať cez jej úplné problémy:

Definícia 6.4 (Polynomiálna hierarchia IV.). *Platí:*

- $\mathsf{\Sigma_k^P} = \{L \mid L \leq_{\mathrm{m}}^{\mathrm{P}} \Sigma_k\textsc{Sat}\}$,
- $\mathsf{\Pi_k^P} = \{L \mid L \leq_{\mathrm{m}}^{\mathrm{P}} \Pi_k\textsc{Sat}\}$.

Viac problémov úplných pre jednotlivé úrovne polynomiálnej hierarchie nájde čitateľ v kompendiu Schaefer a Umans (2002a) a Schaefer a Umans (2002b). Medzi inými, problémy, ktoré sme si predstavili v úvode, sú úplné pre svoje prirodzené triedy:

- Min-Dnf je $\mathsf{\Sigma_2^P}$-úplný,
- 1Lta-Grammar-Inequivalence je $\mathsf{\Sigma_2^P}$-úplný,
- 3-Coloring-Extension je $\mathsf{\Pi_2^P}$-úplný a
- VC-Dimension je $\mathsf{\Sigma_3^P}$-úplný.

6.3 Orákulá

Piata ekvivalentná definícia je cez orákulá. Vezmime si napríklad $\exists\forall\textsc{Sat}$. Pre danú formulu

$$\exists x_1, \ldots, x_m \underbrace{\forall y_1, \ldots, y_n : \phi(x, y)}_{\text{je } \phi_x(y) \text{ tautológia?}}$$

chceme zistiť, či je pravdivá. To je ekvivalentné otázke, či existujú $x_1, \ldots, x_m$ také, že formula $\phi_x(y)$, kde dosadíme hodnoty $x_1, \ldots, x_m$, ale $y_1, \ldots, y_n$ ostávajú voľné premenné, je tautológia. Môžeme teda písať

$$\exists x \forall y : \phi(x, y) \in \exists\forall\textsc{Sat} \iff \text{existuje } x \text{ také, že } \phi_x(y) \in \textsc{Taut}.$$

Ak by sme teda mali nedeterministický Turingov stroj a pridali mu orákulum, ktoré dokáže riešiť Taut, tak takýto stroj dokáže riešiť $\exists\forall\textsc{Sat}$ v polynomiálnom čase: nedeterministicky si tipne x a orákula sa spýta na ϕ_x. Z toho vyplýva, že $\mathsf{\Sigma_2^P} \subseteq \mathsf{NP}^{\textsc{Taut}}$.

Platí to aj naopak? Vieme každý výpočet $\mathsf{NP}^{\textsc{Taut}}$ simulovať Σ_2-strojom v polynomiálnom čase? Na prvý pohľad máme problém: nemôžeme len tak odsimulovať prvú odpoveď orákula v $\forall$-stavoch a následne sa vrátiť do $\exists$-stavov, simulovať nedeterministický stroj, druhú odpoveď zase simulovať v $\forall$-stavoch, atď. K dispozícií máme iba dve alternácie!

Finta je, že výpočet $\mathsf{NP}^{\textsc{Taut}}$ budeme *celý* simulovať v $\exists$-stavoch, pričom všetky odpovede orákula si natipujeme. Ak tipujeme, že odpoveď orákula na i-tu otázku $\phi_i \in \textsc{Taut}$ je Nie, hneď aj natipujeme ohodnotenie, pri ktorom formula nie je splnená a overíme ho. Otázky, kde tipujeme odpoveď Áno, teda formule ϕ_i, ktoré tipujeme, že *sú* tautológie, si zapíšeme a odložíme na neskôr. Všetky ich otestujeme naraz až *na konci* v $\forall$-stavoch. Akceptujeme práve vtedy, ak simulovaný stroj akceptuje a zároveň všetky tipnuté odpovede boli správne.

Teda $\mathsf{NP}^{\textsc{Taut}} = \Sigma_2^{\mathsf{P}}$. Zároveň sme ako bonus dokázali, že trieda $\mathsf{NP}^{\textsc{Taut}}$ ostane rovnaká, aj keď povolíme len jedinú otázku na orákulum TAUT – všetky formule sa totiž dajú (po premenovaní premenných) skombinovať (zANDovať) do jednej otázky.

Pri orákulách je jedno, či máme orákulum pre jazyk alebo jeho komplement (nie je nič jednoduchšie ako spýtať sa orákula a následne odpoveď znegovať). Preto

$$\Sigma_2^{\mathsf{P}} = \mathsf{NP}^{\textsc{Taut}} = \mathsf{NP}^{\textsc{Sat}} = \mathsf{NP}^{\mathsf{NP}} = \mathsf{NP}^{\mathsf{coNP}}.$$

Podobne

$$\Pi_2^{\mathsf{P}} = \mathsf{coNP}^{\textsc{Sat}} = \mathsf{coNP}^{\mathsf{NP}} = \mathsf{coNP}^{\mathsf{coNP}} = \mathsf{co}(\mathsf{NP}^{\mathsf{NP}}).$$

Definícia 6.5 (Polynomiálna hierarchia V.). *Triedy* Σ_k^{P} *a* Π_k^{P} *definujeme induktívne:*

- $\Sigma_0^{\mathsf{P}} = \Pi_0^{\mathsf{P}} = \mathsf{P}$,
- $\Sigma_{k+1}^{\mathsf{P}} = \mathsf{NP}^{\Sigma_k^{\mathsf{P}}}$ *a* $\Pi_{k+1}^{\mathsf{P}} = \mathsf{co}\Sigma_{k+1}^{\mathsf{P}} = \mathsf{coNP}^{\Sigma_k^{\mathsf{P}}} = \mathsf{coNP}^{\Pi_k^{\mathsf{P}}}$.

Okrem toho môžeme definovať $\Delta_{k+1}^{\mathsf{P}} = \mathsf{P}^{\Sigma_k^{\mathsf{P}}} = \mathsf{P}^{\Pi_k^{\mathsf{P}}}$, *pričom* $\Delta_0^{\mathsf{P}} = \Delta_1^{\mathsf{P}} = \mathsf{P}$.

Teda

$$\begin{array}{ccccccccccc} \mathsf{P} & \subseteq & \mathsf{NP} & \subseteq & \mathsf{NP}^{\mathsf{NP}} & \subseteq & \mathsf{NP}^{\mathsf{NP}^{\mathsf{NP}}} & \subseteq & \mathsf{NP}^{\mathsf{NP}^{\mathsf{NP}^{\mathsf{NP}}}} & \subseteq \cdots \\ \| & & \| & & \| & & \| & & \| & \\ \Sigma_0^{\mathsf{P}} & & \Sigma_1^{\mathsf{P}} & & \Sigma_2^{\mathsf{P}} & & \Sigma_3^{\mathsf{P}} & & \Sigma_4^{\mathsf{P}} & \end{array}$$

pričom zátvorkovanie je vždy myslené doprava: $\mathsf{NP}^{\left(\mathsf{NP}^{\left(\mathsf{NP}^{\mathsf{NP}}\right)}\right)}$.

Zjavne $\Delta_{k+1}^{\mathsf{P}} = \mathsf{P}^{\Sigma_k^{\mathsf{P}}} = \mathsf{P}^{\Pi_k^{\mathsf{P}}}$ obsahuje aj Σ_k^{P} aj Π_k^{P} a zjavne naopak $\mathsf{P}^{\Sigma_k^{\mathsf{P}}} \subseteq \mathsf{NP}^{\Sigma_k^{\mathsf{P}}} \cap \mathsf{coNP}^{\Sigma_k^{\mathsf{P}}}$ (pozri obrázok 6.2). $\Delta_{k+1}^{\mathsf{P}}$ sú problémy, ktoré sa dajú polynomiálne *Turingovsky* redukovať na Σ_kSAT:

$$\Delta_{k+1}^{\mathsf{P}} = \{L \mid L \leq_{\mathrm{T}}^{\mathsf{P}} \Sigma_k\textsc{Sat}\} = \{L \mid L \leq_{\mathrm{T}}^{\mathsf{P}} \Pi_k\textsc{Sat}\}.$$

6.4 Paralelizmus

Nakoniec môžeme polynomiálnu hierarchiu definovať pomocou ďalších paralelných modelov ako sú booleovské obvody alebo paralelné RAMy. Polynomiálna hierarchia je ekvivalentná triede problémov, ktoré sa dajú riešiť v *konštantnom* čase, ak máme *exponenciálne* veľké výpočtové zdroje, t.j. exponenciálne veľa hradiel alebo exponenciálne veľa procesorov, ktoré počítajú paralelne.

Presnejšie:

Definícia 6.6 (Polynomiálna hierarchia VI.). *Hovoríme, že trieda booleovských obvodov je DC-uniformná, ak existuje polynomiálny algoritmus, ktorý pre dané* n *vypočíta počet hradiel obvodu pre vstupy dĺžky* n*, pre danú dvojicu* (n, i) *zistí typ* i*-teho hradla (*$\wedge$ *alebo* $\vee$*) a pre dané* (n, i, j) *zistí, či je* i*-te*

hradlo vstup j-teho hradla. Potom PH *je trieda problémov, pre ktoré existuje DC-uniformná trieda monotónnych obvodov (len $\wedge$ a $\vee$ hradlá; ale bez $\neg$ hradiel; na vstupe dostane $x_1, \neg x_1, x_2, \neg x_2, \ldots, x_n, \neg x_n$) exponenciálnej veľkosti, konštantnej hĺbky s neobmedzeným stupňom.*

Definícia 6.7 (Polynomiálna hierarchia VII.). *Uvažujeme model PRAM v CRCW móde. Polynomiálna hierarchia obsahuje práve problémy, ktoré sa dajú riešiť paralelne v konštantnom čase, ak máme k dispozícií exponenciálne veľa procesorov.*

Ekvivalencia týchto dvoch definícií vyplýva zo vzájomnej simulovateľnosti obvodov a PRAMu, ktoré sme spomínali v kapitole o Výpočtových modeloch. Ekvivalenciu s predchádzajúcimi definíciami nechávame ako cvičenie pre čitateľa.

Veta 6.1 (Stockmeyer (1976), Wrathall (1976), Chandra a spol. (1981), Vinay a spol. (1990)). *Všetky uvedené definície polynomiálnej hierarchie I.–VII. sú ekvivalentné.*

Úlohy

- Dokážte, že ak $\Sigma_k^P \subseteq \Pi_k^P$, potom $\Sigma_k^P = \Pi_k^P$.
- Dokážte, že ak $\Sigma_k^P = \Sigma_{k+1}^P$, potom $\Sigma_k^P = \Pi_k^P$.
- Dokážte, že VI. definícia PH cez DC-uniformné obvody je ekvivalentná predošlým.
- Dokážte, že EXP je trieda jazykov s DC-uniformnými obvodmi exponenciálnej veľkosti.

Literatúra

Chandra, Ashok K, Dexter C Kozen, a Larry J Stockmeyer. 1981. "Alternation." *Journal of the ACM (JACM)* 28(1), s. 114–133.

Schaefer, Marcus a Christopher Umans. 2002a. "Completeness in the polynomial-time hierarchy: Part I: A compendium." *SIGACT news* 33(3), s. 32–49.

—. 2002b. "Completeness in the polynomial-time hierarchy: Part II." *SIGACT News* 33(4), s. 22–36.

Stockmeyer, Larry J. 1976. "The polynomial-time hierarchy." *Theoretical Computer Science* 3(1), s. 1–22.

Vinay, V, H Venkateswaran, a CE Veni Madhavan. 1990. "Circuits, pebbling and expressibility." In *Proceedings Fifth Annual Structure in Complexity Theory Conference*. IEEE, s. 223–230.

Wrathall, Celia. 1976. "Complete sets and the polynomial-time hierarchy." *Theoretical Computer Science* 3(1), s. 23–33.

Kapitola 7

Vzťahy s ostatnými triedami

S polynomiálnou hierarchiou sme do rúk dostali oveľa presnejšie „pravítko“, ktorým môžeme merať problémy medzi P a PSPACE. Doteraz sme na stupnici mali iba P, NP a PSPACE, ale pridali sme ďalších nekonečne veľa medzistupňov. V tejto kapitole týmto pravítkom primeriame k iným triedam – k neuniformným obvodom a pravdepodobnostným algoritmom.

7.1 Kolaps polynomiálnej hierarchie

Predpokladáme, že polynomiálna hierarchia je *striktná*, tzn. $\mathsf{P} \subset \mathsf{NP} \subset \Sigma_2^\mathsf{P} \subset \Sigma_3^\mathsf{P} \subset \cdots$ a podobne $\mathsf{P} \subset \mathsf{coNP} \subset \Pi_2^\mathsf{P} \subset \Pi_3^\mathsf{P} \subset \cdots$, pričom každá ďalšia trieda je *ostro* väčšia a obsahuje ťažšie a ťažšie problémy.

Aké sú iné možnosti? Mohlo by sa napríklad stať, že $\mathsf{P} = \mathsf{NP}$, ale ďalej je už hierarchia striktná? Môže sa stať, že $\Sigma_0^\mathsf{P} \subset \Sigma_1^\mathsf{P} = \Sigma_2^\mathsf{P} \subset \Sigma_3^\mathsf{P} = \Sigma_4^\mathsf{P} \subset \cdots$ ($\Sigma_\mathsf{i}^\mathsf{P} \neq \Sigma_{\mathsf{i}+1}^\mathsf{P}$ pre párne i a $\Sigma_\mathsf{i}^\mathsf{P} = \Sigma_{\mathsf{i}+1}^\mathsf{P}$ pre nepárne i)?? A je možné, že $\mathsf{NP} \subset \mathsf{coNP}$, lebo univerzálny kvantifikátor je v nejakom zmysle „silnejší“ ako existenčný?

Nie, ukážeme si, že sú len dve prípustné možnosti:

1) polynomiálna hierarchia je nekonečná a striktná; každá ďalšia úroveň dokáže ostro viac, $\Sigma_\mathsf{i}^\mathsf{P} \subset \Sigma_{\mathsf{i}+1}^\mathsf{P}$, a triedy $\Sigma_\mathsf{i}^\mathsf{P}$ a $\Pi_\mathsf{i}^\mathsf{P}$ sú neporovnateľné pre všetky i, alebo

2) existuje k také, že po k-tu úroveň je hierarchia striktná, $\mathsf{P} = \Sigma_0^\mathsf{P} \subset \Sigma_1^\mathsf{P} \subset \Sigma_2^\mathsf{P} \subset \cdots \subset \Sigma_\mathsf{k}^\mathsf{P}$, ale na k-tej úrovni $\Sigma_\mathsf{k}^\mathsf{P} = \Pi_\mathsf{k}^\mathsf{P}$ a všetky zvyšné úrovne skolabujú: $\Sigma_\mathsf{k}^\mathsf{P} = \Pi_\mathsf{k}^\mathsf{P} = \Sigma_{\mathsf{k}+1}^\mathsf{P} = \Sigma_{\mathsf{k}+2}^\mathsf{P} = \cdots = \mathsf{PH}$.

Skúste si najskôr sami dokázať tieto jednoduché tvrdenia:

- Ak $\Sigma_\mathsf{k}^\mathsf{P} \subseteq \Pi_\mathsf{k}^\mathsf{P}$, potom $\Sigma_\mathsf{k}^\mathsf{P} = \Pi_\mathsf{k}^\mathsf{P}$.
- Ak $\Sigma_\mathsf{k}^\mathsf{P} = \Sigma_{\mathsf{k}+1}^\mathsf{P}$, potom $\Sigma_\mathsf{k}^\mathsf{P} = \Pi_\mathsf{k}^\mathsf{P}$.

Veta 7.1 (Kolaps PH). *Ak* $\mathsf{P} = \mathsf{NP}$*, tak celá celá polynomiálna hierarchia skolabuje na* $\mathsf{PH} = \mathsf{P}$*. Ak* $\mathsf{NP} = \mathsf{coNP}$*, tak* $\mathsf{PH} = \mathsf{NP}$*.*

Vo všeobecnosti ak $\Sigma_i^\mathsf{P} = \Pi_i^\mathsf{P}$*, tak* $\mathsf{PH} = \Sigma_i^\mathsf{P}$ *a hovoríme, že polynomiálna hierarchia skolabuje na i-tu úroveň.*

■ **Dôkaz.** Indukciou dokážeme, že $\mathsf{P} = \Sigma_i^\mathsf{P} = \Pi_i^\mathsf{P}$ pre všetky i. Báza indukcie $i = 1$ je predpoklad našej vety ($\mathsf{NP} = \Sigma_1^\mathsf{P}$). Všimnime si, že trieda P je uzavretá na komplement a preto ak $\mathsf{P} = \Sigma_i^\mathsf{P}$, tak automaticky $\mathsf{P} = \Sigma_i^\mathsf{P} = \Pi_i^\mathsf{P}$.

Predpokladajme, že $\mathsf{P} = \Sigma_i^\mathsf{P} = \Pi_i^\mathsf{P}$ a ukážme, že potom $\mathsf{P} = \Sigma_{i+1}^\mathsf{P} = \Pi_{i+1}^\mathsf{P}$. Nech $L \in \Sigma_{i+1}^\mathsf{P}$. To znamená, že existuje polynomiálny stroj M taký, že

$$x \in L \iff \exists u_0 \underbrace{\forall u_1 \cdots Q u_i : M(x, u_0, \ldots, u_i) = 1}_{(x,u_0)\in L'}$$

(kde $u_0, \ldots, u_i$ sú polynomiálne dlhé). Definujme jazyk

$$L' = \{(x, u_0) \mid \forall u_1 \cdots Q u_i : M(x, u_0, \ldots, u_i) = 1\}.$$

Ten má o jeden kvantifikátor menej a teda zjavne $L' \in \Pi_i^\mathsf{P}$. Avšak z indukčného predpokladu $\Pi_i^\mathsf{P} = \mathsf{P}$ a teda $L' \in \mathsf{P}$.

Ak však $L' \in \mathsf{P}$, tak existuje deterministický Turingov stroj M', ktorý ho akceptuje v polynomiálnom čase a potom zjavne $L \in \mathsf{NP}$, pretože

$$x \in L \iff \exists u : M'(x, u) = 1.$$

No ale my sme predpokladali, že $\mathsf{P} = \mathsf{NP}$ a preto $L \in \mathsf{P}$. Takto sme ukázali, že *ak* $\mathsf{P} = \Sigma_i^\mathsf{P} = \Pi_i^\mathsf{P}$, *tak* aj ľubovoľný jazyk z Σ_{i+1}^P patrí do P a teda $\mathsf{P} = \Sigma_{i+1}^\mathsf{P} = \Pi_{i+1}^\mathsf{P}$, čím sme dokázali indukčný krok.

Skrátka, ak $\mathsf{P} = \exists \cdot \mathsf{P} = \forall \cdot \mathsf{P}$, znamená to, že operátory $\exists$ a $\forall$ s triedou P „nič nerobia“. Potom napríklad $\Sigma_3^\mathsf{P} = \exists \cdot (\forall \cdot (\exists \cdot \mathsf{P}) = \exists \cdot (\forall \cdot \mathsf{P}) = \exists \cdot \mathsf{P} = \mathsf{P}$.

Druhá časť vety sa ukáže podobne (prenechávame čitateľovi). □

Z toho vyplýva, že napríklad dokázať, že $\mathsf{NP} \neq \mathsf{coNP}$ je ťažšie (alebo aspoň tak ťažké) ako dokázať, že $\mathsf{P} \neq \mathsf{NP}$. Ak by sa nám podarilo dokázať, že $\Sigma_3^\mathsf{P} \neq \Pi_3^\mathsf{P}$, automaticky platí $\mathsf{P} \subset \mathsf{NP} \subset \Sigma_2^\mathsf{P} \subset \Sigma_3^\mathsf{P}$.

V predchádzajúcej kapitole sme si ukázali rôzne Σ_k^P- a Π_k^P-úplné problémy. Má aj celá PH nejaký PH-úplný problém? Skúste si rozmyslieť, že ak by taký existoval, tak PH skolabuje na Σ_k^P pre nejaké k.

7.2 Obvody a polynomiálna hierarchia

Jeden z možných útokov na problém či $\mathsf{P} = \mathsf{NP}$ je pomocou booleovských obvodov. P je trieda jazykov akceptovaných uniformnými obvodmi polynomiálnej veľkosti, avšak väčšina dolných odhadov na veľkosť obvodov uniformitu vôbec nevyužíva a tieto dolné odhady platia aj v neuniformnom modeli.

Núka sa nám teda otázka: aký je vzťah medzi NP a $\mathsf{P/poly}$? Vedeli by sme dokázať, že na riešenie SATu potrebujeme exponenciálne veľký obvod? (Alebo aspoň superpolynomiálne veľký?)

Vieme, že $\mathsf{P} \subseteq \mathsf{P/poly}$, takže

$$\text{ak } \mathsf{P} = \mathsf{NP} \Longrightarrow \mathsf{NP} \subseteq \mathsf{P/poly}.$$

Inými slovami, ak $\mathsf{NP} \not\subseteq \mathsf{P/poly}$, ak sa SAT nedá riešiť obvodmi polynomiálnej veľkosti, potom $\mathsf{P} \neq \mathsf{NP}$. Takže skúšať dokázať, že $\mathsf{P} \neq \mathsf{NP}$ pomocou dolných odhadov na veľkosť obvodov je rozumná možnosť. Nanešťastie, zatiaľ najsilnejší odhad, ktorý vieme dokázať, je že na riešenie SATu treba aspoň $5n+o(n)$ hradiel.

Zaujímavá je však aj otázka, či platí aj opačná implikácia. Je pravda, že

$$\text{ak } \mathsf{NP} \subseteq \mathsf{P/poly} \Longrightarrow \mathsf{P} = \mathsf{NP}?$$

Inými slovami, sú tvrdenia $\mathsf{P} \neq \mathsf{NP}$ a $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ ekvivalentné? Odpoveď je, že zatiaľ nevieme. Avšak vieme dokázať o čosi slabšie tvrdenie. $\mathsf{P} = \mathsf{NP}$ znamená, že celá polynomiálna hierarchia skolabuje na P (na nultú úroveň) – to dokázať nevieme. Avšak vieme, že ak by $\mathsf{NP} \subseteq \mathsf{P/poly}$, tak polynomiálna hierarchia skolabuje na *druhú* úroveň.

Veta 7.2 (Karp (1982)). *Ak* $\mathsf{NP} \subseteq \mathsf{P/poly}$*, tak* $\mathsf{PH} = \Sigma_2^{\mathsf{P}}$.

■ **Dôkaz.** Nech $\mathsf{NP} \subseteq \mathsf{P/poly}$, ukážeme, že Π_2^{P}-úplný problém $\forall\exists$SAT obsahujúci pravdivé formuly tvaru $\forall u \exists v \phi(u,v)$ patrí do Σ_2^{P}. Podľa predpokladu existuje trieda obvodov $\{C_n\}_{n\in\mathbb{N}}$ polynomiálnej veľkosti taká, že C_n ku každému ϕ a u nájde vhodné v, aby $\phi(u,v)$ bola pravda.

Obvod C_n sa dá zapísať pomocou polynomiálneho počtu bitov, takže formula

$$\forall u\ \exists v : \phi(u, \underbrace{v}_{\text{necháme vypočítať obvodom } C_n})$$

sa dá prepísať do tvaru

$$\exists \langle C_n \rangle\ \forall u : \phi(u, \underbrace{C_n(\phi, u)}_{\text{splňujúce } v}),$$

z čoho zjavne vyplýva $\forall\exists\text{SAT} \in \Sigma_2^{\mathsf{P}}$. To, či pre každé u existuje „vhodné“ v rozhodneme tak, že zistíme, či existuje obvod, ktorý ku každému u „vhodné“ v vypočíta. □

Podobné tvrdenia (že ak $\mathcal{C} \subseteq \mathsf{P/poly}$, tak celé $\mathcal{C}$ skolabuje na Σ_2^{P}) sa dajú dokázať pre viacero tried $\mathcal{C}$. Napríklad ak by sa všetky problémy riešiteľné v exponenciálnom čase dali riešiť polynomiálnymi obvodmi, potom $\mathsf{EXP} = \Sigma_2^{\mathsf{P}}$:

Veta 7.3 (Meyer). *Ak* $\mathsf{EXP} \subseteq \mathsf{P/poly}$*, tak* $\mathsf{EXP} = \Sigma_2^{\mathsf{P}}$.

■ **Dôkaz.** Nech $L = L(M) \in \mathsf{EXP}$ a definujme jazyk

$$L_M = \{(x, i, j) \mid \text{v } i\text{-tej konfigurácii výpočtu na vstupe } x \text{ je na } j\text{-tej pozícii } 1\}.$$

Zrejme ak celý výpočet trvá exponenciálne dlho, tak v exponenciálnom čase viem aj výpočet odsimulovať, stopnúť v i-tom kroku a pozrieť sa na j-tu pozíciu konfigurácie, teda zjavne $L_M \in \mathsf{EXP}$.

Keby $\mathsf{EXP} \subseteq \mathsf{P/poly}$, potom aj $L_M \in \mathsf{P/poly}$, čo znamená, že celý exponenciálne dlhý výpočet stroja M vieme zakódovať do polynomiálne veľkého obvodu. Jazyk L potom vieme rozhodovať takto:

$$\begin{aligned} x \in L &\iff \exists\text{exponenciálne dlhý akceptačný výpočet stroja } M \\ &\iff \exists\text{polynomiálny } \langle C \rangle\ \forall i, j : \textsc{Verify}_M(C, x, i, j), \end{aligned}$$

kde $\textsc{Verify}_M$ je algoritmus, ktorý skontroluje, že obvod C kóduje akceptačný výpočet stroja M. Konkrétne pre j-tu pozíciu v i-tej konfigurácii zistí, či zodpovedá j-tej pozícii v predošlej konfigurácii, či je v počiatočnej konfigurácii na vstupe x a či je v poslednej konfigurácii povedzme hlava úplne vľavo (normálny tvar) a v akceptačnom stave. Aj napriek tomu, že konfigurácie sú exponenciálne dlhé, každé jedno políčko závisí len od konštantne veľa predošlých, indexy i, j majú polynomiálnu dĺžku a čo sa nachádza na pozícii (i, j) vieme vďaka obvodu C vypočítať v polynomiálnom čase. A tak sme ukázali, že keby $\mathsf{EXP} \subseteq \mathsf{P/poly}$, ľubovoľný jazyk $L \in \mathsf{EXP}$ vieme zapísať v tvare „$\exists\forall$“ a teda patrí do Σ_2^{P}. ukážeme, že $L \in \Sigma_2^{\mathsf{P}}$. □

Skeptický čitateľ možno začne pochybovať, či dôkazy takýchto viet majú vôbec zmysel. Niektorí informatici takéto vety nazývajú ironicky „vety typu keby prasatá vedeli písakať, tak kone by vedeli lietať“. No ale my veríme, že prasatá nevedia písakať. A tiež, že kone nevedia lietať. Tak načo to je dobré?

V skutočnosti máme v teórii zložitosti veľa tvrdení a hypotéz, ktoré nevieme dokázať. Na začiatok je teda možno užitočné ozrejmiť si, či sú niektoré hypotézy ekvivalentné, alebo je niektorá silnejšia, či jedna vyplýva z druhej, atď. Konkrétne, ak veríme, že polynomiálna hierarchia je striktná, aké dôsledky z toho vyplývajú?

Druhý dôvod je môžno akási viera, že zhromažďovaním ďalších a ďalších dôsledkov sa tieto tvrdenia budú dať pospájať do niečoho užitočnejšieho. A to sa naozaj deje. Existuje viacero príkladov, ktoré dokazujú užitočnosť podobných viet (pozri napríklad kapitoly Zložitosť počítania a Derandomizácia a dolné odhady).

My si zatiaľ ukážeme len jeden absolútne neužitočný, zato filozoficky veľmi zaujímavý až podivný dôsledok predchádzajúcej vety. Vidíte, čo je na ňom čudné?

Dôsledok 7.1. *Ak* $\mathsf{P} = \mathsf{NP}$*, tak* $\mathsf{EXP} \not\subseteq \mathsf{P/poly}$.

■ **Dôkaz.** Ak $\mathsf{P} = \mathsf{NP}$, tak $\mathsf{P} = \Sigma_2^{\mathsf{P}}$. Keby $\mathsf{EXP} \subseteq \mathsf{P/poly}$, tak $\mathsf{EXP} = \Sigma_2^{\mathsf{P}} = \mathsf{P}$. To je ale spor s časovou hierarchiou. □

Zvláštnosť tohto tvrdenia spočíva v tom, že zatiaľčo doteraz sme mali výsledky typu „ak vieme riešiť takéto problémy, tak vieme riešiť aj takéto“, „ak je toto ľahké, potom aj toto je ľahké“, podľa tejto vety ak je SAT ľahký, potom EXP obsahuje *ťažké* problémy (ktoré sa nedajú riešiť polynomiálnymi obvodmi).

Na záver tejto sekcie si ešte ukážme že už na druhej úrovni polynomiálnej hierarchie vieme nájsť problémy, ktoré nedokážu riešiť polynomiálne obvody veľkosti n^k (pre fixné k). V úlohe z kapitoly 5 bolo treba ukázať existenciu problémov v $\mathsf{PSPACE} - \mathsf{SIZE}(n^k)$. To bola rozcvička. Teraz tento výsledok dramaticky zlepšíme až na Σ_2^P namiesto PSPACE.

Samozrejme, radi by sme našli nejaký problém z NP, ktorý je ťažký aj pre obvody, ale toto hľadanie je zatiaľ beznádejné. Nevieme dokázať ani len superlineárny dolný odhad.

Veta 7.4 (Kannan (1982)). *Pre každé k existuje jazyk z $\Sigma_2^\mathsf{P} \cap \Pi_2^\mathsf{P}$, ktorý nie je v $\mathsf{SIZE}(n^k)$.*

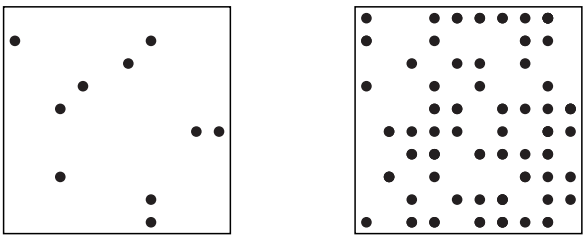
Pozor, táto veta *nehovorí*, že existuje problém v $\Sigma_2^\mathsf{P} \cap \Pi_2^\mathsf{P} - \mathsf{P/poly}$. Dobre si rozmyslite, aký je v tom rozdiel.

■ **Dôkaz.** Uvažujme jazyk $L \in \mathsf{SIZE}(n^{k+1}) - \mathsf{SIZE}(n^k)$, ktorý je počítaný lexikograficky najmenším obvodom – takýto jazyk existuje vďaka neuniformnej hierarchii. Potom $L \in \Sigma_4^\mathsf{P}$. Vetu „$x \in L$, kde L je jazyk z $\mathsf{SIZE}(n^{k+1}) - \mathsf{SIZE}(n^k)$ s lexikograficky najmenším obvodom“ vieme totiž zapísať pomocou 4 vnorených striedavých kvantifikátorov. Potrebujeme overiť, že

1. existuje obvod C veľkosti n^{k+1} taký, že
2. neexistuje ekvivalentný obvod veľkosti n^k – čiže každý malý obvod c sa na nejakom slove líši od C
3. a zároveň C je najmenší taký – tzn. pre každý lexikograficky menší obvod existuje obvod veľkosti n^k, ktorý akceptuje rovnaký jazyk (resp. rovnaké slová dĺžky n)
4. takýto obvod C akceptuje x.

$$
\begin{aligned}
x \in L \iff & \exists\text{obvod } C \text{ taký, že} \\
& \Big(\big[\forall\text{obvod } c \text{ veľkosti } n^k \ \exists\text{slovo } y : C(y) \neq c(y)\big] \\
& \wedge \big[\forall\text{lexikograficky menší obvod } D \\
& \quad \exists\text{obvod } d \text{ veľkosti } n^k, \forall z : D(z) = d(z)\big] \\
& \wedge C(x) = 1 \Big)
\end{aligned}
$$

(V tejto formulácii máme ešte pomiešané logické spojky a kvantifikátory, dôležité však je, že v najhlbšej vetve sa striedajú 4 kvantifikátory: $\exists C \forall D \exists d \forall z$. Podľa pravidiel predikátovej logiky vieme túto vetu prepísať do tzv. prenexného tvaru



a) ak $x \notin L(M)$, S_x je malá

b) ak $x \in L(M)$, S_x je veľká

Obr. 7.1: Štvorec znázorňuje priestor $\{0,1\}^m$ všetkých m-bitových reťazcov (náhodné bity, ktoré používa BPP-algoritmus M). S_x je množina náhodných reťazcov $r \in \{0,1\}^m$, na ktorých $M(x,r)$ akceptuje. Ako rozlíšime prípady a) S_x je malá, vs. b) S_x je veľká?

$\exists C : \forall c, D : \exists y, d : \forall z : \phi$. Alebo Σ_4^{P}-stroj akceptujúci L môže po natipovaní C prepnúť do univerzálneho stavu a paralelne overiť všetky tri $\wedge$-vetvy v zátvorke.)

No dobre, takto sme dokázali, že $L \in \Sigma_4^{\mathsf{P}} - \mathsf{SIZE}(n^k)$, ale veta sľubovola Σ_2^{P}, nie?

Zvyšok dôkazu je srandovný: ďalej budeme postupovať podľa toho, či sa SAT dá riešiť obvodmi veľkosti n^k. Odpoveď síce nepoznáme, ale v oboch prípadoch naša veta platí:

- Ak SAT $\notin \mathsf{SIZE}(n^k)$, tak SAT je jazyk, ktorý hľadali (SAT $\in \Sigma_2^{\mathsf{P}} \cap \Pi_2^{\mathsf{P}}$).
- Naopak, ak SAT $\in \mathsf{SIZE}(n^k)$, tak PH podľa Karp-Liptona skolabuje a tým pádom sa jazyk L, ktorý je na štvrtej úrovni, prepadne na druhú. $L \in \mathsf{PH}$, ale ak SAT $\in \mathsf{P/poly}$, tak $\mathsf{PH} = \Sigma_2^{\mathsf{P}} \cap \Pi_2^{\mathsf{P}}$. □

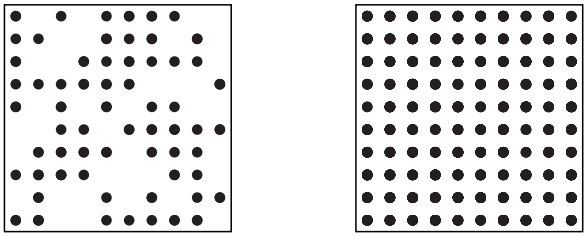
Pekný oťob, nie? Intuicionisti by protestovali.

7.3 Náhodnosť a polynomiálna hierarchia

Nakoniec sa pozrime ešte na vzťah medzi polynomiálnou hierarchiou a BPP. Hoci veríme, že náhodnosť pri riešení problémov až tak nepomáha a $\mathsf{BPP} = \mathsf{P}$, nevieme dokázať ani len $\mathsf{BPP} \subseteq \mathsf{NP}$. Vieme však aspoň dokázať, že BPP patrí pod druhú úroveň:

Veta 7.5 (Sipser, Gács, Lautemann (1983)). $\mathsf{BPP} \subseteq \Sigma_2^{\mathsf{P}} \cap \Pi_2^{\mathsf{P}}$.

■ **Dôkaz.** Nech M je BPP stroj pre jazyk L, ktorý na n-bitovom vstupe x použije m náhodných bitov a mýli sa len s exponenciálne malou pravdepodobnosťou. Označme S_x množinu náhodných reťazcov, na ktorých M akceptuje. Vieme, že ak $x \in L$, tak S_x je „veľká“ ($\geq (1 - 2^{-n})2^m$) a ak $x \notin L$, tak S_x je „malá“ ($\leq 2^{m-n}$), pozri obr. 7.1. Ako tieto dva prípady odlíšiť?



a) ak $x \notin L(M)$, S nezaberá celý priestor

b) ak $x \in L(M)$, S zaberá celý priestor

Obr. 7.2: Nech $S = \bigcup_{i=1}^{k} S_x \oplus t_i$ je množina S_x posunutá o náhodné vektory. V prípade a) S nezaberá celý priestor, v prípade b) je $S = \{0,1\}^m$ s veľkou pravdepodobnosťou.

Pod *posunutím* S o t budeme myslieť množinu $S \oplus t = \{r \oplus t \mid r \in S\}$. Dokážeme, že ak $x \in L$, tak existujú $t_1, \ldots, t_k$ také, že posunutia S_x o t_i dokopy pokryjú všetky m-bitové reťazce, teda $S = \bigcup_{i=1}^{k} S_x \oplus t_i = \{0,1\}^m$ a pre $x \notin L$ sa to nedá, pozri obr. 7.2. Potom $x \in L \iff \exists t_1, \ldots, t_k \forall r\ M$ akceptuje na jednom z $t_i \oplus r$, teda $L \in \Sigma_2^{\mathsf{P}}$; z uzavretosti na komplement vyplýva $L \in \Pi_2^{\mathsf{P}}$.

Nech $k = 2m/n$. Ak S_x je malá, tak $S = \bigcup_i S_x \oplus t_i$ bude zaberať najviac k-násobne viac a $|S| \leq k \times |S_x| = 2m/n \times 2^m/2^n \ll 2^m$, takže S určite nepokryje všetky reťazce pre žiadnu voľbu posunutí.

Naopak, ak S_x je veľká, ukážeme, že náhodné posunutia pokryjú $\{0,1\}^m$ s nenulovou pravdepodobnosťou a teda *nejaké existujú*. Toto je princíp tzv. pravdepodobnostnej metódy: ak $\Pr_x[\neg E(x)] < 1$, potom $\exists x : E(x)$.

Vezmime si nejaký konkrétny bod $r \in \{0,1\}^m$. Náhodné posunutia ho *nepokryjú* práve vtedy, keď

$$r \notin \bigcup_i S_x \oplus t_i \iff \forall i : r \oplus t_i \notin S_x.$$

Pravdepodobnosť tejto udalosti je najviac $(2^{-n})^k$, pretože $r \oplus t_i$ sú náhodné body z $\{0,1\}^m$. Pravdepodobnosť, že existuje bod, ktorý nie je pokrytý je najviac $2^m \times$toľko (všetkých bodov je 2^m; union bound), čo je najviac $2^{m-nk} = 2^{-m} \ll 1$, teda nie vždy. Takže s nenulovou pravdepodobnosťou bude pokrytý každý bod a preto existuje konkrétna sada posunutí, ktorá to dosiahne. □

Úlohy

- V súčasnosti nevieme, či $\mathsf{BPP} \subseteq \mathsf{NP}$. Napriek tomu dokázateľne platí, že ak $\mathsf{P} = \mathsf{NP}$, tak $\mathsf{P} = \mathsf{BPP}$. Prečo?

- Predpokladajme, že PH je striktná. Vyplýva z toho, že $\mathsf{PH} \subset \mathsf{PSPACE}$?

- Nech $C = (C_1, \ldots, C_n)$ je n-tica obvodov, kde C_i pracuje na vstupoch dĺžky i. Hovoríme, že C rieši SAT, ak pre každú formulu ϕ dĺžky $i \leq n$ je $C_i(\phi) = 1 \iff \phi \in \text{SAT}$. Dokážte, že problém SATCIRCUITVERIFICATION, pre dané C povedať, či C rieši SAT, je v coNP.

Literatúra

Kannan, Ravi. 1982. "Circuit-size lower bounds and non-reducibility to sparse sets." *Information and control* 55(1-3), s. 40–56.

Karp, Richard. 1982. "Turing machines that take advice." *Enseign. Math.* 28, s. 191–209.

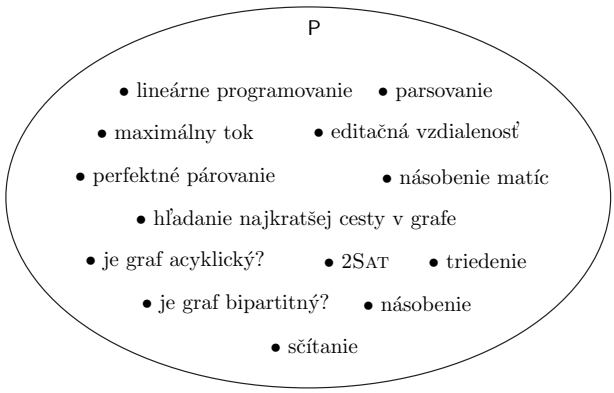
Lautemann, Clemens. 1983. "BPP and the polynomial hierarchy." *Information Processing Letters* 17(4), s. 215–217.

Časť III

Logické obvody (Dajú sa problémy paralelizovať?)

Úvod

V tejto časti sa pokúsime trochu si priblížiť, ako vyzerá svet „pod P“ – budeme sa zaoberať úlohami, ktoré sú riešiteľné v polynomiálnom čase. Je to veľmi bohatý svet, ktorý obsahuje všakovaké problémy, počnúc jednoduchou aritmetikou (sčítanie, násobenie, delenie, najväčší spoločný deliteľ), cez triedenie a vyhľadávanie, grafové algoritmy (prehľadávanie, hľadanie najkratšej cesty, maximálneho toku), optimalizačné úlohy, kde zafunguje dynamické programovanie (napríklad výpočet editačnej vzdialenosti, či parsovanie) až po násobenie matíc a lineárne programovanie.



Tieto problémy by sme mohli skúsiť ďalej roztriediť podľa ich časovej zložitosti. Nás však v tejto časti budú zaujímať ďalšie aspekty zložitosti: paralelizmus a malá pamäť.

Konkrétnejšie:

1. Dajú sa tieto problémy riešiť *rýchlo paralelne*?

2. Dajú sa počítať s (extrémne) *malou pamäťou*?

3. A aký je vôbec vzťah medzi paralelným časom a pamäťou?

Paralelizmus

Majme úlohu, ktorá sekvenčne trvá čas T. Ak si na ňu vezmeme počítač s p procesormi, v najlepšom prípade sa bude dať práca rozložiť rovnomerne medzi všetky výpočtové jednotky a paralelne ju vyriešime v čase T/p. Inými slovami, to najlepšie, v čo môžeme dúfať, je p-násobné zrýchlenie. Naopak, ak každý procesor počíta t krokov, celková *„práca“* bude $p \times t$ a zhruba v tomto čase (+možno nejaká réžia navyše v závislosti od modelu) vieme celý výpočet odsimulovať aj sekvenčne.

Za „praktické“ považujeme iba modely, ktorých hardware (napríklad počet procesorov) je len polynomiálne veľký – a teda môžeme dúfať v nanajvýš polynomiálne zrýchlenie. Z toho napríklad vyplýva, že exponenciálne ťažké problémy budú stále potrebovať exponenciálny čas. Na druhej strane, aj polynomiálne dlhé paralelné výpočty sa budú ešte stále dať odsimulovať aj sekvenčne v polynomiálnom čase.

Ktoré paralelné algoritmy teda budeme považovať za *rýchle*?

Ukazuje sa, že veľa zaujímavých problémov sa dá riešiť paralelne v čase $O(\log n)$ alebo $O(\log^2 n)$, navyše „rozumné“ paralelné modely sa vedia navzájom simulovať tak, že k časovej zložitosti možno pribudne dáky ten logaritmus navyše. Preto je veľmi prirodzené definovať ako „efektívne paralelizovateľné“ problémy tie, ktoré sa dajú riešiť *v polylogaritmickom čase* (v čase polynomiálnom od $\log n$, t.j. $O(\log^k n)$, kde k je konštanta) s polynomiálne veľa procesormi. Pre túto triedu sa ujal názov NC (Nick's class) na počesť Nicholasa Pippengera. Je to robustná trieda, je jedno, či ju definujeme cez CRCW alebo EREW PRAM, alebo budeme používať tak, ako v nasledujúcich kapitolách, obvody polynomiálnej veľkosti a polylogaritmickej hĺbky – či už s obmedzeným, alebo neobmedzeným stupňom.

V nasledujúcej kapitole sa budeme zaoberať otázkou, ktoré problémy z P patria do NC. Otázka, či $\mathsf{NC} \stackrel{?}{=} \mathsf{P}$ je stále otvorená; podobne ako pri $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ však vieme vytypovať tie „najťažšie“ a „najsekvenčnejšie“ problémy v P – ak by sa tie dali paralelizovať, potom sa dá všetko.

Malá pamäť

Druhá téma, ktorou sa budeme v tejto časti zapodievať je: Ktoré problémy sa dajú riešiť s malou – konkrétne logaritmickou – pamäťou? To sú triedy L a NL, s ktorými sme sa už stretli. Zjavne ak výpočet používa len logaritmickú pamäť, existuje len polynomiálne veľa možných konfigurácii, do ktorých sa výpočet môže dostať. Môžeme teda vytvoriť graf všetkých konfigurácii a zistiť, či sa z počiatočnej dá dostať do akceptačnej. Celé to zvládneme v polynomiálnom čase, takže zjavne $\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P}$. Otázky, či $\mathsf{L} \stackrel{?}{=} \mathsf{NL}$, alebo či $\mathsf{NL} \stackrel{?}{=} \mathsf{P}$ sú stále otvorené.

Paralelizmus vs. pamäť

Ak sú ale dobre paralelizovateľné problémy (trieda NC) aj problémy riešiteľné s malou pamäťou (L či NL) zjavne pod P, ďalšia prirodzená otázka znie: Aký je vzťah medzi týmito triedami? Aký je vzťah medzi paralelizmom a malou pamäťou?

Podľa Tézy o paralelných výpočtoch sú čas na „rozumnom“ paralelnom modeli a priestor na „rozumnom“ sekvenčnom modeli zhruba ekvivalentné (až na polynomiálny faktor). V predchádzajúcej časti sme si zase ukázali, že AP = PSPACE – problémy riešiteľné paralelne v polynomiálnom čase sú práve tie problémy, ktoré sa dajú riešiť v polynomiálnej pamäti. Ďalšie, presnejšie súvislosti medzi paralelizmom, pamäťou a alternovaním si ukážeme v nasledujúcej kapitole.

Kapitola 8

Paralelizmus

V tejto kapitole nás bude zaujímať, ktoré úlohy sa dajú riešiť rýchlo *paralelne*.

Príklad 1: Chceme zistiť, či je počet jednotiek na vstupe párny.

Príklad 2: Súčet dvoch prirodzených čísiel.

Príklad 3: Súčin dvoch prirodzených čísiel.

Príklad 4: Existuje cesta medzi dvoma vrcholmi v danom grafe?

Príklad 5: Dajú sa regulárne jazyky rozpoznávať paralelne? A čo bezkontextové?

Dajú sa tieto problémy riešiť v logaritmickom čase? Alebo aspoň *polylogaritmickom* čase (t.j. $O(\log^k n)$ pre fixné k)? A naopak, vedeli by sme o nejakých problémoch dokázať, že sa nedajú rýchlo paralelizovať?

8.1 Triedy NC a AC

Definícia 8.1. NC *je trieda jazykov akceptovaných (log-space) uniformnými boolovskými obvodmi polynomiálnej veľkosti a polylogaritmickej hĺbky*

Špeciálne NC^k *je trieda jazykov akceptovaných „dostatočne“ uniformnými*[1] *boolovskými obvodmi polynomiálnej veľkosti, hĺbky* $O(\log^k n)$. *Hradlá* $\wedge$ *a* $\vee$ *tu majú vždy 2 vstupy.* NC *je potom zjednotenie týchto tried pre všetky* k*:* $\mathsf{NC} = \bigcup_k \mathsf{NC}^k$.

Podobne definujeme AC^k *s tým rozdielom, že povolíme hradlá* $\wedge$ *a* $\vee$ *s neobmedzeným počtom vstupov.* $\mathsf{AC} = \bigcup_k \mathsf{AC}^k$.

[1] Na tomto mieste musíme upozorniť, že pre $k \geq 2$ je logspace uniformita v poriadku, avšak pre veľmi malé triedy ako AC^0 či NC^1, ktoré sú *pod* L si treba dávať pozor na detaily. Ak by sme tieto triedy definovali cez logspace uniformitu, už len samotný log-space stroj, ktorý má obvod vygenerovať, by dokázal vypočítať viac ako samotné obvody. Rôzne definície uniformity a ich prípadné ekvivalencie nebudeme rozoberať a záujemcov odkážeme na články Ruzzo (1981) a Barrington a spol. (1990).

Ako ukázali Stockmeyer a Vishkin (1984), AC^k je práve trieda problémov riešiteľná na CRCW PRAM v čase $O(\log^k n)$ s polynomiálne veľa procesormi.

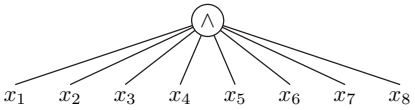
Triviálne platí, že $\mathsf{NC}^k \subseteq \mathsf{NC}^{k+1}$, $\mathsf{AC}^k \subseteq \mathsf{AC}^{k+1}$ a taktiež $\mathsf{NC}^k \subseteq \mathsf{AC}^k$. Navyše každé hradlo s polynomiálne veľa vstupmi sa dá simulovať binárnym stromom logaritmickej hĺbky (pozri obrázok 8.1) a teda $\mathsf{AC}^k \subseteq \mathsf{NC}^{k+1}$. Teda

$$\mathsf{AC}^0 \subseteq \mathsf{NC}^1 \subseteq \mathsf{AC}^1 \subseteq \mathsf{NC}^2 \subseteq \cdots \subseteq \mathsf{NC}$$

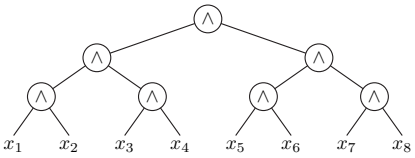
a

$$\mathsf{NC} = \bigcup_k \mathsf{NC}^k = \bigcup_k \mathsf{AC}^k = \mathsf{AC}.$$

Nevieme, či je táto hierarchia striktná a či $\mathsf{AC}^k \subsetneq \mathsf{NC}^k \subsetneq \mathsf{AC}^{k+1}$ pre všetky k, avšak keby $\mathsf{NC}^k = \mathsf{NC}^{k+1}$, tak celá NC hierarchia skolabuje na k-tu úroveň: $\mathsf{NC} = \mathsf{NC}^k$.



(a) AC obvody môžu používať hradlá s neobmedzeným počtom vstupov.



(b) Tie dajú simulovať NC obvodom logaritmickej hĺbky.

Obr. 8.1

8.2 Rýchle paralelné algoritmy

Veta 8.1. *Sčítanie prirodzených čísiel je v AC^0, teda vieme ho spočítať paralelne v konštantom čase pomocou hradiel s neobmedzeným stupňom:* $\textsc{Add} \in \mathsf{AC}^0$.

■ **Dôkaz.** Označme binárny zápis čísel na vstupe $a_n \cdots a_1 a_0$, $b_n \cdots b_1 b_0$. Jediný problém je, ako paralelne vypočítať prenos do vyššieho rádu (označme $c_n \cdots c_1 c_0$). Skúsme si uvedomiť, ako sa prenos do vyššieho rádu šíri na konkrétnom príklade:

```
1 0 0 1 0 1 1 1 0 1 0 1
1 0 0 0 1 1 0 0 1 1 0 0
  ↑           ↑
```

Ako vieme, že na mieste ľavej šípky bude prenos +1 z nižšieho rádu? Platí, že $c_i = 1$ práve vtedy, keď pre nejaké $j < i$ je $a_j = b_j = 1$ (tu sa prenosy začnú, pozri pravú šípku v príklade) a zároveň odvtedy pre všetky $j < k < i$ je aspoň jedno z a_k, b_k jednotka (čo zabezpečí, že prenos pokračuje), teda

$$c_i = \bigvee_{j<i} a_j \wedge b_j \wedge \left(\bigwedge_{j<k<i} a_k \vee b_k \right).$$

Súčet je potom $s_i = a_i \oplus b_i \oplus c_i$. □

Čo násobenie?

Školácky algoritmus: prvé číslo vynásobíme každou cifrou druhého čísla (toto vieme triviálne paralelne v konštantnom čase) a posunuté výsledky sčítame. Otázka znie, ako sčítame n n-bitových čísel? Mohli by sme ich sčítavať po dvoch podľa stromu logaritmickej hĺbky. Takto však dostaneme iba algoritmus v NC^2, respektíve v AC^1. Dá sa to lepšie?

Veta 8.2 (Avizienis (1961), Wallace (1964)). *Násobenie prirodzených čísel dokážeme paralelne v logaritmickom čase od počtu cifier:* Mult $\in \mathsf{NC}^1$.

■ **Dôkaz.**

Finta je, že 3 sčítania vieme (paralelne) v konštantom čase zredukovať na 2 sčítania. Súčet n čísel teda vieme v $O(1)$ zredukovať na súčet $\frac{2}{3}n$ čísel. Po $\log_{3/2} n$ iteráciach nám ostanú už len posledné dve čísla, ktoré sčítame algoritmom uvedeným vyššie.

Redukciu 3 sčítaní na 2 si ukážme na príklade:

```
    0 1 0 0 1 1
+   0 1 1 0 0 1
+   1 0 1 0 0 1
-----------------
    1 0 0 0 1 1     súčty mod 2
+ 0 1 1 0 0 1       prenosy do vyššieho rádu
```

Jednoducho nečakáme, či nastane prenos z nižších rádov. Namiesto toho sčítame len cifry v každom stĺpci zvlášť a z prenosov do vyššieho rádu vyskladáme nové číslo: Napríklad v poslednom stĺpci máme $1+1+1 = 3 = (11)_2$, teda súčet $1+1$ prenos do vyššieho rádu – ten zapíšeme o riadok nižšie vľavo. V predposlednom stĺpci nečakáme na prenos, iba sčítame $1+0+0 = 1$, teda súčet 1 a prenos 0 do vyššieho rádu. Takto dostaneme dve čísla – súčty mod 2 a prenosy do vyšších rádov. Súčet horných troch čísel sme zredukovali na súčet dvoch.

Dôležité je, že keď sčítame 3 binárne cifry, súčet bude $[0..3]$, čiže sa dá zapísať ako dvojciferné číslo v dvojkovej sústave. Podobne by sme mohli zredukovať sčítanie 7 čísel na súčet 3-och, alebo súčet 31 čísel na súčet 5-tich.

Iný spôsob je použitím tzv. *redundantných* číselných sústav (pozri napríklad Parhami (2010)). Klasicky reprezentujeme čísla v sústave so základom z s ciframi $d_i \in \{0, 1, \ldots, z-1\}$ ako

$$(d_{n-1} \cdots d_2 d_1 d_0)_z = \sum_{i=0}^{n-1} d_i \times z^i.$$

Môžeme však uvažovať cifry s väčšieho rozsahu – dokonca môžeme uvažovať záporné cifry. Nevýhoda takejto reprezentácie je, že nie je jednoznačná (ak chceme napríklad zistiť, či sa dve čísla rovnajú, musíme ich previesť do nejakého

normálneho tvaru). Na druhej strane, väčšia voľnosť nám umožňuje sčítať dve čísla *bez* prenosov do vyššieho rádu.

Ukážme si to na príklade v desiatkovej sústave s ciframi v rozsahu [0..18]. Napríklad číslo $(11, 9, 17, 5, 12, 18)_{10} = 11 \cdot 10^5 + 9 \cdot 10^4 + 17 \cdot 10^3 + 5 \cdot 100 + 12 \cdot 10 + 18 = (1207638)_{10}$ (zápis je nejednoznačný). Dve takéto čísla môžeme sčítať nasledovne:

	11	9	17	5	12	18	*redundantný zápis: 10-tková sústava*
+	6	12	9	10	8	18	*s ciframi [0..18]*
	17	21	26	15	20	36	*medzivýsledok má cifry [0..36]*
	7	1	6	5	0	16	*rozložíme ich na 10×[0..2]+[0..16]*
+	1	2	2	1	2	2	*(tu sú prenosy do vyššieho rádu) a sčítame*
	1	9	3	7	7	2	16 *cifry výsledku budú opäť [0..18]*

Ak cifry sčítame paralelne po stĺpcoch, dostaneme medzisúčet s ciframi [0..36]. Keďže $36 = 20 + 16$, tieto cifry sa dajú zapísať ako dvojciferné čísla, kde prvá cifra je [0..2] a druhá cifra [0..16]. Medzivýsledok teda rozdelíme na dve čísla s ciframi v menšom rozsahu. Ich sčítaním paralelne po stĺpcoch dostaneme výsledok opäť s ciframi v rozsahu [0..18]. Takýto algoritmus sa dá implementovať dokonca v NC^0, teda NC-obvodom konštantnej hĺbky. Sčítanie n čísiel potom vieme v logaritmickej hĺbke. Na záver ešte musíme previesť číslo z redundantného zápisu do klasického – a vykonať „odložené“ prenosy. Toto vieme v AC^0, respektíve v NC^1 ako pri sčítaní dvoch čísel.

Skúste porozmýšľať, v ktorých číselných sústavách (pri ktorých základoch a rozsahoch cifier) vieme implementovať sčítanie v NC^0 bez prenosu do vyššieho rádu. □

OK, dá sa to ešte lepšie? Dá sa násobenie spočítať v AC^0?
Odpovieme v nasledujúcej kapitole.

Veta 8.3. *Zistiť, či existuje cesta medzi dvoma vrcholmi v (orientovanom) grafe sa dá v* AC^1.

■ **Dôkaz.** Pomocou boolovského násobenia matíc.[2] Majme maticu susednosti A, tzn. $A_{ij} = 1$, ak je v grafe hrana medzi i—j, inak $A_{ij} = 0$. Potom A^2 má jednotku na pozícii ij práve vtedy, keď sa z i do j vieme dostať na dva kroky. Všeobecnejšie, majme matice susednosti A a B nad tou istou množinou vrcholov. Predstavme si, že jednotky v A sú červené hrany a jednotky v B modré hrany. Potom súčin $A \cdot B$ je matica C, kde $C_{ij} = \bigvee_k A_{ik} \wedge B_{kj}$. Teda $C_{ij} = 1$, ak sa

[2] Súvis medzi násobením matíc a rôznymi úlohami o sledoch grafov je technika, ktorú sa oplatí poznať. Vo všeobecnosti môžeme definovať matice nad nejakým okruhom $(K, \oplus, \otimes)$ a ich súčin $C_{ij} = \bigoplus_k (A_{ik} \otimes B_{kj})$. Rôzne problémy sa dajú riešiť násobením/umocňovaním matíc nad vhodným okruhom. Napríklad pri boolovskom násobení je $(A^k)_{ij} = 1$, ak existuje sled dĺžky k medzi i a j. Pri „klasickom“ násobení nad celými číslami je $(A^k)_{ij}$ počet sledov dĺžky k medzi i a j. No a v praxi je veľmi zaujímavý okruh $(\mathbb{R} \cup \{\infty\}, \min, +)$ a tzv. min-plus násobenie: $C_{ij} = \min_k(A_{ik} + B_{kj})$. Potom $(A^k)_{ij}$ je dĺžka najkratšieho sledu medzi i a j.

dá dostať z i do j na dva kroky (cez nejaký vrchol k), najskôr po červenej a následne po modrej hrane.

$A^k = A \cdot A \cdots A$ je matica, kde $A_{ij} = 1$, ak sa dá z i do j dostať na *presne* k krokov. Ak matici pridáme jednotky na diagonálu (matica $I \vee A$), je to ako keby sme pridali ku každému vrcholu slučku – každý sled má potom možnosť použitím slučky ostať vo vrchole a $(I \vee A)^k = I \vee A \vee A^2 \vee \cdots \vee A^k$ je matica, kde na pozícií ij je jednotka, ak sa dá z i do j dostať na *najviac* k krokov.

Boolovský súčin matíc vieme zjavne vypočítať v AC^0. Opakovaným umocňovaním na druhú vieme v $\log n$ krokoch vypočítať tzv. reflexívno-tranzitívny uzáver matice $A^* = (I \vee A)^n$, kde $A^*_{ij} = 1$, ak sa dá z i dostať do j. □

8.3 Vzťah medzi malým priestorom a paralelizmom

Problém hľadania cesty v grafe je významný aj pre triedy s logaritmickým priestorom. Tento problém zrejme patrí do NL: ak existuje cesta medzi danými dvoma vrcholmi, stačí postupne natipovať postupnosť vrcholov a skontrolovať, že medzi každými dvoma po sebe idúcimi vrcholmi je hrana. Ak má graf n vrcholov, každý vrchol je označený $\log n$ bitmi a pri kontrole si stačí naraz pamätať iba dva vrcholy (netreba celú cestu), na čo stačí $O(\log n)$ pamäte.

Čo je však dôležitejšie:

Veta 8.4. *Problém zistiť, či existuje cesta medzi dvoma vrcholmi v danom orientovanom grafe je* NL*-úplný pri logspace redukcii.*

■ **Dôkaz.** Nech $L \in \mathsf{NL}$ je jazyk akceptovaný NTS M v logaritmickom priestore a pre daný vstup x si predstavme graf, kde vrcholy sú všetky možné konfigurácie stroja M a hrany $C \to C'$ vedú medzi konfiguráciami, ak $C \vdash_M C'$. Bez újmy na všeobecnosti môžeme predpokladať, že existuje jedinečná konfigurácia, v ktorej M akceptuje (tesne pred akceptovaním vymaže obsah pások a vráti hlavy na začiatočné políčka). Otázku, či M akceptuje x, môžeme redukovať na hľadanie cesty v grafe konfigurácií medzi počiatočnou a akceptačnou konfiguráciou.

Ostáva rozmyslieť si, že túto redukciu vieme vypočítať v logaritmickom priestore. Jedna konfigurácia (pozície hláv, stav a obsahy pások) sa dá zapísať ako reťazec dĺžky $O(\log n)$, takže jednoducho prejdeme cez všetky reťazce logaritmickej dĺžky a tie, ktoré kódujú konfigurácie, vypíšeme ako vrcholy; následne prejdeme cez všetky dvojice konfigurácií (generujeme všetky reťazce a kontrolujeme, či kódujú konfigurácie) a skontrolujeme, či sa M dostane z prvej do druhej na jeden krok (či sedia obsahy pások, či sa hlavy hýbu podľa δ-funkcie) – tieto dvojice vypíšeme ako hrany. Nakoniec triviálne dopíšeme počiatočnú a akceptačnú konfiguráciu. □

Nevieme, či $\mathsf{L} \neq \mathsf{NL}$. Ak by sa hľadanie cesty dalo riešiť deterministicky v logaritmickom priestore, potom by $\mathsf{L} = \mathsf{NL}$.

Tu mi nedá nespomenúť, hoci je to malá odbočka, veľmi príbuzný problém a síce hľadanie cesty v *neorientovanom* grafe. Pre tento problém existuje veľmi jednoduchý randomizovaný algoritmus: začni v začiatočnom vrchole a pohybuj sa náhodne, v každom vrchole si hoď kockou a vydaj sa pozdĺž náhodnej hrany; ak si po veľa veľa krokoch stále nedošiel do cieľa, skús začať znovu zo začiatku a tento pokus veľakrát zopakuj; ak sa ani po veľa pokusoch nepodarilo dostať do cieľa, prehlás, že cesta neexistuje. Dá sa dokázať, že tento algoritmus naozaj funguje v zmysle, že ak cesta existuje, pravdepodobnosť, že ju nenájdeme môžeme stlačiť pod ľubovoľne (exponenciálne) malú hranicu. A teda hľadanie cesty v neorientovanom grafe je v RL.

Čo je ešte fascinujúcejšie, tento algoritmus sa podarilo derandomizovať a cesta sa dá nájsť deterministicky v logaritmickom priestore, teda v L. Dajú sa všetky problémy v RL derandomizovať? Platí $\mathsf{L} = \mathsf{RL}$? Nevieme. Vie aj pri hľadaní cesty v orientovanom grafe pomôcť náhoda? Je $\mathsf{RL} = \mathsf{NL}$? Nevieme. Každopádne rovnaký algoritmus, že skúšam náhodnú prechádzku pre orientované grafy vo všeobecnosti nefunguje. Na rozdiel od neorientovaných grafov sa dajú zostrojiť vstupy, kde by sme sa takto dostali do cieľa iba s exponenciálne malou pravdepodobnosťou.

Avšak vráťme sa späť k otázke NL vs. NC. Z toho, že hľadanie cesty je NL-úplný problém riešiteľný v NC vyplýva, že celé $\mathsf{NL} \subseteq \mathsf{NC}$. Presnejšie, plati:

Veta 8.5. $\mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{AC}^1 \subseteq \mathsf{NC}^2$.

■ **Dôkaz.** Výsledok obvodu vieme vyhodnotiť rekurzívnym algoritmom, pričom si pamätáme iba číslo hradla, ktoré práve vyhodnocujeme a $O(1)$ informácie pre každé hradlo „nad ním“, teda ak si obvod predstavíme ako binárny strom, pamäť je úmerná ceste od nejakého vrcholu ku koreňu (pre OR-hradlá sú buď „zatiaľ všetky vstupy 0“, alebo narazíme na jednotku a výsledok tohto hradla je 1; podobne pre ANDy). Obvody logaritmickej hĺbky teda vieme vyhodnotiť v logaritmickom priestore ($\mathsf{NC}^1 \subseteq \mathsf{L}$).

NL-úplný problém hľadania cesty v orientovanom grafe vieme vyriešiť v AC^1 počítaním tranzitívneho uzáveru matice opakovaným umocňovaním na druhú. □

8.4 Charakterizácia cez alternáciu

Už sme si dokázali, že NC je niekde medzi NL a P:

$$\mathsf{NL} \subseteq \mathsf{NC} \subseteq \mathsf{P}.$$

Ak si však spomenieme na predchádzajúcu časť o alternácii, $\mathsf{P} = \mathsf{AL}$, teda polynomiálny čas vieme charakterizovať cez logaritmický priestor + alternáciu.

Žiadna alternácia je L, 1 alternácia je NL, veľa alternácie je zase P. A NC je niekde medzi. Z čoho prirodzene vyplýva otázka: Vedeli by sme NC charakterizovať presnejšie cez logaritmický priestor a alternáciu?

Odpoveď je: Áno. NC je v skutočnosti to isté ako logaritmický priestor + *polylogaritmicky veľa* alternácie.

Definícia 8.2. *Definujme triedu* $\mathsf{STA}(s(n), t(n), Xa(n))$ *jazykov akceptovaných TS, ktoré počítajú v priestore* $O(s(n))$ *a zároveň v čase* $O(t(n))$ *a* $a(n)$*-krát alternujú, pričom alternácie začínajú v stave* $X \in \{\Sigma, \Pi\}$ *(ak na tom nezáleží,* X *vynecháme; ak je niektorý zdroj neobmedzený, označíme ho* $*$*).*

Napríklad

$$\begin{aligned}
\mathsf{L} &= \mathsf{STA}(\log, *, 0) \\
\mathsf{P} &= \mathsf{STA}(*, \mathrm{poly}, 0) = \mathsf{STA}(\log, *, *) = \mathsf{AL} \\
\mathsf{NP} &= \mathsf{STA}(*, \mathrm{poly}, \Sigma_1) \\
\Pi_k^{\mathsf{P}} &= \mathsf{STA}(*, \mathrm{poly}, \Pi_k) \\
\mathsf{PSPACE} &= \mathsf{STA}(\mathrm{poly}, *, 0) = \mathsf{STA}(*, \mathrm{poly}, *) = \mathsf{AP}
\end{aligned}$$

Veta 8.6 (Ruzzo (1981)). $\mathsf{NC} = \mathsf{STA}(\log, *, \mathrm{polylog})$*. Presnejšie, pre* $k \geq 1$ *platí:* $\mathsf{AC}^k = \mathsf{STA}(\log, *, \log^k)$.[3]

■ **Dôkaz.** $\subseteq$: Upravíme na monotónny obvod so vstupmi $x, \overline{x}$ a simulujeme: $\vee$ existenčnými a $\wedge$ univerzálnymi stavmi. Počet alternácií je najviac hĺbka obvodu, t.j. polylog n. Obvod nemáme celý, ale generujeme si ho za behu.

$\supseteq$: Pomocou násobenia matíc; logspace stroj má polynomiálne veľa možných konfigurácií. Nech $\mathrm{typ}(\alpha)$ je typ stavu – $\exists$ alebo $\forall$; na vstupe x uvažujme matice S_x, T_x, M_x také, že

- $S_x(\alpha, \beta) \iff \alpha \vdash \beta \wedge \mathrm{typ}(\alpha) = \mathrm{typ}(\beta)$ a
- $T_x(\alpha, \beta) \iff \alpha \vdash \beta \wedge \mathrm{typ}(\alpha) \neq \mathrm{typ}(\beta)$;

nech $M_x = S_x^* T_x$. Potom $M_x(\alpha, \beta) = 1$, ak $\alpha \vdash^* \gamma \vdash \beta$, pričom počas celého výpočtu $\alpha \vdash^* \gamma$ sme v stave rovnakého typu ($\exists/\forall$), až v poslednom kroku $\gamma \vdash \beta$ alternujeme. Rozsekajme strom výpočtu ATS na jednotlivé $\exists/\forall$ úrovne, vrámci ktorých je stroj v stave rovnakého typu (úrovní je polylog n). Konfigurácia α na úrovni $i+1$ je akceptačná, ak existuje, resp. pre všetky β na úrovni i je $M_x(\alpha, \beta) = 1$ a β je akceptačná konfigurácia. Definujme vektor b_i: $b_i(\alpha) = 1$ ak α je akceptačná konfigurácia na i-tej úrovni. Ak použijeme konvenciu, že $\forall$-konfigurácie bez nasledujúceho kroku akceptujú a $\exists$-konfigurácie odmietajú, môžeme položiť $b_0 = 0$. Ďalej pre $\exists$-úrovne vypočítame $b_{i+1} = M_x b_i$ a pre $\forall$-úrovne $b_{i+1} = \neg(M_x(\neg b_i))$. Akceptujeme, ak $b_{\log^c n}(\sigma) = 1$ pre štartovaciu konfiguráciu σ. □

[3]Pozor, veta asi neplatí pre $k = 0$, ak $\mathsf{AC}^0 \neq \mathsf{NL}$: Totiž logspace hierarchia $\mathsf{STA}(\log, *, O(1))$ nie je nič iné ako trieda NL (prečo?). Na druhej strane, dá sa dokázať, že $\mathsf{AC}^0 = \mathsf{LH}$, tzv. *logtime* hierarchia – problémy v LH sa dajú vyriešiť v logaritmickom čase s konštantným počtom alternácií. (Pri definícii logtime predpokladáme, že stroj má špeciálnu indexovaciu pásku, kde napíše číslo i a hlava sa presunie na i-ty symbol vstupu.)

Dôsledok 8.1.

$$\begin{array}{ccc} \mathsf{NL} & = & \mathsf{STA}(\log, *, \Sigma_1) \\ \cap\mathrel{|} & & \cap\mathrel{|} \\ \mathsf{AC}^1 & = & \mathsf{STA}(\log, *, \log) \\ \cap\mathrel{|} & & \cap\mathrel{|} \\ \mathsf{NC} & = & \mathsf{STA}(\log, *, \mathrm{polylog}) \\ \cap\mathrel{|} & & \cap\mathrel{|} \\ \mathsf{P} & = & \mathsf{STA}(\log, *, *) \end{array}$$

8.5 Paralelizmus a parsovanie*

Veta 8.7. *Regulárne jazyky sa dajú akceptovať v* NC^1.

■ **Dôkaz.** Prenechávame čitateľovi. □

Veta 8.8 (Ruzzo (1980), Ruzzo (1981), Rytter (1986)). *Bezkontextové jazyky vieme akceptovať v* AC^1.

■ **Náznak dôkazu.** Uvažujme gramatiku v Chomského normálnom tvare. Označme w vstupné slovo a $w_{i,j}$ podreťazec $w_i w_{i+1} \cdots w_{j-1}$.

Využijeme charakterizáciu z vety 8.6, že AC^1 sú práve problémy riešiteľné alternujúcim TS v logaritmickej pamäti s logaritmickým počtom alternácií. Algoritmus budeme prezentovať ako hru Alice a Boba, kde Alica (to sú existenčné stavy) sa snaží dokázať, že $\sigma \Rightarrow_G^* w$ a Bob (univerzálne stavy) sa snaží prichytiť Alicu pri klamstve.

Alica na ťahu si vyberie trojicu (α, i, j), čím tvrdí, že

$$\sigma \Rightarrow_G^* w_{1,i}\alpha w_{j,n} \qquad \text{a} \qquad \alpha \Rightarrow_G^* w_{i,j}$$

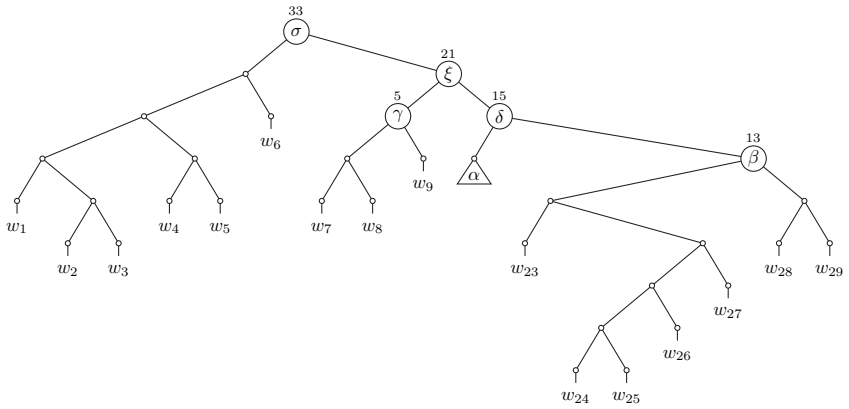
Bob si vyberie, ktorému z týchto dvoch tvrdení neverí a ktoré mu má Alica dokázať. Hra sa končí, keď sa dostane do stavu, že výsledok možno overiť podľa pravidiel gramatiky a vstupného slova – napríklad $\alpha \Rightarrow_G^* \beta\gamma$ alebo $\alpha \Rightarrow_G^* w_i$ – ak príslušné pravidlo patrí do gramatiky, Alica vyhrá, v opačnom prípade vyhrá Bob.

Zjavne, ak $w \in L(G)$, Alica vie pravdivo odpovedať na každú výzvu a vyhrá a naopak, ak sa slovo v gramatike nedá odvodiť, v každom ťahu je aspoň jedno z Alicinych tvrdení nepravdivé a Bob ju vie nachytať. Ostáva nám dokázať, že existuje stratégia, pri ktorej Alica vyhrá v logaritmickom čase a pozície hry sa budú dať reprezentovať v logaritmickej pamäti.

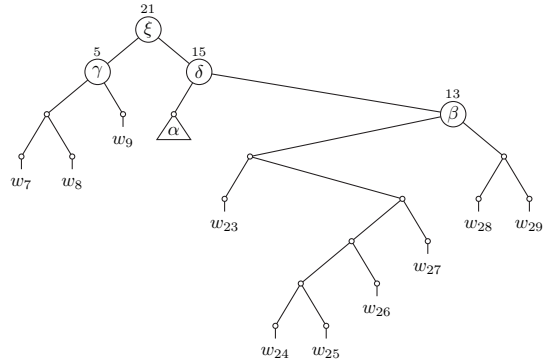
Skúste si rozmyslieť, že ľubovoľný binárny strom vieme zmazaním hrany rozdeliť na dve časti tak, že obe časti majú $\frac{1}{3}$ až $\frac{2}{3}$ všetkých vrcholov. Ak bude Alica vyberať takéto vrcholy, hra skončí po zhruba $\log_{3/2} n$ ťahoch.

Ostáva vyriešiť problém logaritmickej pamäte: ak by si Alica vyberala vždy nové podstromy a Bob by si vyberal vždy hornú časť, vo všeobecnosti by sa hra dostala do stavu, že

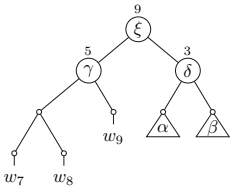
$$\sigma \Rightarrow_G^* w_{i_1,i_2}\ \alpha_1\ w_{i_3,i_4}\ \alpha_2\ w_{i_5,i_6}\ \alpha_3 \cdots \alpha_k\ w_{i_{2k+1},i_{2k+2}}$$



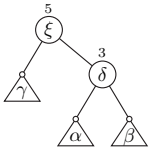
(a) V prvom ťahu Alica zvolila trojicu $(\alpha, 10, 23)$ a Bob si vybral hornú časť; Alica teda potrebuje dokázať, že $\sigma \Rightarrow_G^* w_{1,10}\alpha w_{23,30}$. Keďže pod ξ je 21 vrcholov, čo je niekde medzi $\frac{1}{3}$ a $\frac{2}{3}$ všetkých vrcholov, Alica zvolí trojicu $(\xi, 7, 30)$. Bob si môže vybrať, či chce vidieť, ako $\sigma \Rightarrow_G^* w_{1,7}\xi$, alebo $\xi \Rightarrow_G^* w_{7,10}\alpha w_{23,30}$.



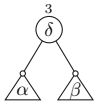
(b) Bob si vybral druhú možnosť. Keďže pod β je 13 vrcholov, čo je niekde medzi $\frac{1}{3}$ a $\frac{2}{3}$ zostávajúcich vrcholov, Alica zvolí trojicu $(\beta, 10, 30)$. Bob si môže vybrať, či chce vidieť, ako $\xi \Rightarrow_G^* w_{7,10}\alpha\beta$, alebo $\beta \Rightarrow_G^* w_{23,30}$.



(c) Bob si vybral prvú možnosť. Alica: $(\gamma, 7, 10)$.



(d) Bob si vybral hornú časť stromu. Alica: $(\delta, 10, 30)$.



(e) Bob si vybral dolnú časť. Po overení, že $\delta \to \alpha\beta$ je pravidlo gramatiky Alica vyhráva.

Obr. 8.2: Hra Alice a Boba na parsovacom strome pre slovo w. Tu je jeden možný priebeh hry.

Každý index $i_1, \ldots, i_{2k+2}$ zaberá logaritmickú pamäť; ak hra trvá len logaritmický počet ťahov, budeme mať $k = O(\log n)$ medzier a teda len $O(\log n)$ indexov, čo je však stále až $O(\log^2 n)$ pamäte.

Skúste si však rozmyslieť, že ak máme binárny strom a tri vrcholy α, β, γ, pričom žiadny z nich nie je spoločným predkom ostatných dvoch, potom vieme zvoliť vrchol γ, ktorý je predok práve dvoch vrcholov z $\{\alpha, \beta, \gamma\}$. Inými slovami, nech si Bob vyberie ktorúkoľvek časť, ostanú nám len 2 medzery (dva neterminály).

Alica môže hrať tak, že vždy, keď sa dostane do stavu s troma neterminálmi, zvolí vrchol, aby ich počet zredukovala na dva a v stave s dvoma a menej neterminálmi (to je v najhoršom prípade každý druhý ťah) volí vrchol, ktorý zmenší počet vrcholov najviac na 2/3. Takto hra po $O(\log n)$ ťahoch skončí a zároveň v každej pozícii si treba pamätať najviac 3 neterminály a 8 indexov, čo dokážeme v $O(\log n)$ pamäti. □

Trieda bezkontextových jazykov CFL sa len ťažko dá porovnávať s ostatnými triedami v NC, keďže nie je uzavretá na logspace redukcie. V teórii zložitosti sa preto uvažuje trieda LOGCFL problémov, ktoré sa dajú redukovať na nejaký bezkontextový jazyk:

$$\mathsf{LOGCFL} = \{A \mid \exists L \in \mathsf{CFL} : A \leq_{\mathrm{m}}^{\log} L\}.$$

Zrejme $\mathsf{L} \subseteq \mathsf{LOGCFL}$ a podľa predchádzajúcej vety $\mathsf{LOGCFL} \subseteq \mathsf{AC}^1$. Všimnite si, že zatiaľčo Alica mala na výber veľa možností, Bob mal zakaždým najviac dve. Z toho sa dá odvodiť, že jazyky z LOGCFL sa dajú počítať tzv. *semi-unbounded* obvodmi, kde ANDy majú stupeň 2, ale ORy môžu mať neobmedzený stupeň. Trieda takýchto obvodov logaritmickej hĺbky je SAC^1. Venkateswaran (1991) dokázal, že LOGCFL je presne SAC^1.

8.6 Neparalelizovateľné?

Ktoré problémy nevieme efektívne paralelizovať? Ktoré problémy nepatria do NC?

V súčasnosti nevieme dokázať, že $\mathsf{NC} \neq \mathsf{P}$ a teda je možné, že každý problém z P sa dá počítať v polylogaritmickom čase polynomiálne veľkými obvodmi. Avšak podobne ako sme si pri otázke $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ vytypovali tie najťažšie problémy v NP (NP-úplné problémy), vieme si vytypovať tie najťažšie problémy v P, tzv. P-úplné problémy. (Samozrejme, potrebujeme jemnejšiu ako polynomiálnu redukciu – napr. NC alebo logspace redukciu.)

Problém je P-ťažký, ak sa naň dá redukovať každý problém v P. Ak tento problém navyše patrí do P, hovoríme, že je P-úplný. Ak by ľubovoľný P-úplný problém patril do NC, tak celé $\mathsf{P} = \mathsf{NC}$.

Veta 8.9. *Vyhodnotenie daného boolovského obvodu na danom vstupe (tzv.* CVP *– circuit value problem) je* P*-úplný problém, dokonca pri* AC^0*-redukcií.*

Ďalšie P*-úplné problémy sú*

- HornSat – *problém splniteľnosti pre Hornove formule (v CNF, každá klauzula má najviac jeden pozitívny literál)*
- Dfs – *daný je graf a dva vrcholy u, v; ak na grafe spustíme prehľadávanie do hĺbky (pričom zoznam susedov prehľadávame vo fixnom poradí danom na vstupe), ktorý vrchol z dvojice u, v navštívime ako prvý?*
- MaxFlow – *daný je ohodnotený graf, vrcholy s, t a číslo f; existuje s-t-tok veľkosti aspoň f?*
- LP – *lineárne programovanie: existuje pre danú maticu A a vektor b riešenie nerovníc $Ax \leq b$, $x > 0$ v racionálnych číslach?*
- CfgMem – *$w \in L(G)$? pre danú bezkontextovú gramatiku G a slovo w (pozor, toto je úplne iný problém, ako rozhodovať jazyk $L(G)$ pre fixnú bezkontextovú gramatiku)*
- IteratedMod – *dané $a, b_1, \ldots, b_n \in \mathbb{Z}$; je $((\cdots((a \bmod b_1) \bmod b_2)\cdots) \bmod b_n) = 0$?*

Obsiahly zoznam ďalších známych P-úplných problémov čitateľ nájde v Greenlaw a spol. (1991) alebo v knižke Greenlaw a spol. (1995).

Úlohy

- Uvažujme redundantnú číselnú sústavu so základom z a s ciframi v rozsahu $[-\alpha..\beta]$ (kde $\alpha + \beta + 1 \geq z$) ako vo vete 8.2. Pre ktoré z, α, β viete navrhnúť algoritmus sčítania bez prenosov do vyššieho rádu (ktorý sa dá implementovať v NC^0)?
- TC obvody (*threshold circuits*) môžu okrem $\neg$ a neobmedzených $\wedge$ a $\vee$ používať navyše hradlo Maj (s neobmedzeným počtom vstupov), ktoré vráti 1, ak sa *väčšina* vstupov rovná 1. Analogicky definujeme triedy TC^k polynomiálnej veľkosti a hĺbky $O(\log^k n)$. Dokážte, že $\mathsf{ACC}^0 \subseteq \mathsf{TC}^0 \subseteq \mathsf{NC}^1$. Všeobecne, $\mathsf{ACC}^k \subseteq \mathsf{TC}^k \subseteq \mathsf{NC}^{k+1}$.
- Zostrojte triedu AC obvodov konštantnej hĺbky a *logaritmickej* veľkosti, ktorá akceptuje všetky binárne reťazce, ktoré majú aspoň 2 jednotkové bity, t.j. obvody, ktoré akceptujú jazyk $L = \{x \mid \#_1(x) \geq 2\} = \{x \mid \exists i \neq j : x_i = x_j = 1\}$.
- Dokážte, že regulárne jazyky sa dajú rozoznávať paralelne v logaritmickom čase: $\mathsf{REG} \subseteq \mathsf{NC}^1$.
- Dokážte, že NC^1 je presne trieda problémov riešiteľná boolovskými formulami polynomiálnej veľkosti. (Formula sa dá definovať ako obvod, kde výstupný stupeň každého hradla je najviac 1.)

- Majme alternujúci Turingov stroj, ktorý pracuje v logaritmickom priestore. Jeho výpočet môže byť až polynomiálne dlhý a preto celý strom výpočtov môže byť až exponenciálne veľký. Dokážte, že ak máme ATS pracujúci v logaritmickom priestore (a možno až polynomiálnom čase), ale jeho strom výpočtov je len polynomiálne veľký, dá sa simulovať na ATS v logaritmickom priestore a $O(\log^2 n)$ čase. To znamená, že výpočty s malou pamäťou a malým stromom výpočtov (ak sú výpočty dlhé, nie je ich veľa) sa dajú dobre paralelizovať. Všeobecne platí, že ak máme ATS, ktorý pracuje v priestore $s(n) \geq \log n$ a jeho strom výpočtov má veľkosť $z(n)$, tak existuje ekvivalentný ATS, ktorý pracuje v priestore $O(s(n))$ a čase $O(s(n) \log z(n))$. Rýchly paralelný algoritmus pre CFL z vety 8.8 je špeciálny prípad tohto tvrdenia.

Literatúra

Avizienis, Algirdas. 1961. "Signed-digit number representations for fast parallel arithmetic." *IRE Transactions on electronic computers* 10(3), s. 389–400.

Barrington, David A Mix, Neil Immerman, a Howard Straubing. 1990. "On uniformity within NC^1." *Journal of Computer and System Sciences* 41(3), s. 274–306.

Greenlaw, Raymond, H James Hoover, a Walter L Ruzzo. 1991. *A compendium of problems complete for* P. Citeseer.

Greenlaw, Raymond, H James Hoover, Walter L Ruzzo a spol. 1995. *Limits to parallel computation: P-completeness theory.* Oxford University Press on Demand.

Parhami, Behrooz. 2010. *Computer arithmetic*, roč. 20. Oxford university press.

Ruzzo, Walter L. 1980. "Tree-size bounded alternation." *Journal of Computer and System Sciences* 21(2), s. 218–235.

—. 1981. "On uniform circuit complexity." *Journal of Computer and System Sciences* 22(3), s. 365–383.

Rytter, Wojciech. 1986. "On the complexity of parallel parsing of general context-free languages." *Theoretical computer science* 47, s. 315–321.

Stockmeyer, Larry a Uzi Vishkin. 1984. "Simulation of parallel random access machines by circuits." *SIAM Journal on Computing* 13(2), s. 409–422.

Venkateswaran, Hari. 1991. "Properties that characterize LOGCFL." *Journal of Computer and System Sciences* 43(2), s. 380–404.

Wallace, Christopher S. 1964. "A suggestion for a fast multiplier." *IEEE Transactions on electronic Computers* (1), s. 14–17.

Kapitola 9

Parita a obvody konštantnej hĺbky

V predchádzajúcej kapitole sme zaviedli AC/NC-hierarchiu obvodov:

$$\mathsf{AC}^0 \subseteq \mathsf{NC}^1 \subseteq \mathsf{AC}^1 \subseteq \mathsf{NC}^2 \subseteq \mathsf{AC}^2 \subseteq \cdots \subseteq \mathsf{NC}.$$

V tejto kapitole sa budeme venovať tej najmenšej, hoci o nič menej zaujímavej triede AC^0 – obvodmi konštantnej hĺbky a polynomiálnej veľkosti.

Budú nás zaujímať otázky ako:
Dokážu AC^0 obvody

Príklad 1: spočítať paritu n bitov?

Príklad 2: sčítanť dve prirodzené čísla?

Príklad 3: vynásobiť dve prirodzené čísla?

Príklad 4: vynásobiť dve boolovské matice?

Príklad 5: spočítať majoritu n bitov (ktorý bit sa vyskytuje častejšie)?

Ako sme videli v predchádzajúcej kapitole, odpoveď na otázky 2 a 4 je kladná. Tu si naopak dokážeme, že odpoveď na 1, 3 a 5 je záporná. Násobenie prirodzených čísel ani majorita n bitov sa v AC^0 implementovať nedá. Kameň úrazu spočíva v tom, že, ako si ukážeme ochvíľu, v AC^0 sa nedá ani len zistiť, či je na vstupe párny alebo nepárny počet jednotiek, t.j. pre daný binárny reťazec x nevieme spočítať $\sum x_i \pmod 2 = \bigoplus x_i$.

Samozrejme, parita sa dá jednoducho spočítať v NC^1 – v logaritmickej hĺbke, stačí implementovať $a \oplus b = (a \wedge \neg b) \vee (\neg a \wedge b)$ a z takýchto $\oplus$ hradiel vyskladať binárny strom nad x.

Parita je teda príklad problému, ktorý vieme riešiť v NC^1, dokonca v konštantnej pamäti (je to regulárny jazyk), ale nevieme ho riešiť v AC^0. Teda hoci

nevieme, či je celá AC/NC hierarchia striktná, *vieme* to dokázať aspoň pre tú najspodnejšiu úroveň:

$$\mathsf{AC}^0 \neq \mathsf{NC}^1.$$

Ak chceme paritu spočítať v konštantnej hĺbke, povedzme d, polynomiálny obvod nestačí. Aký veľký obvod potrebujeme? Prvý výsledok, že $\textsc{Parity} \notin \mathsf{AC}^0$ dokázali Furst a spol. (1984). Výsledky sa postupne zlepšovali a vďaka Håstad (1986) dnes poznáme veľmi presné horné aj dolné odhady:

Veta 9.1 (Håstad (1986), Håstad (1987)). *Logický obvod hĺbky d, ktorý počíta paritu, musí mať veľkosť aspoň $2^{\Omega(\sqrt[d-1]{n})}$, presnejšie viac ako $2^{\sqrt[d-1]{n/10^d}}$ pre dostatočne veľké n Tento odhad je takmer optimálny, keďže parita sa dá vypočítať v hĺbke d obvodom veľkosti $n2^{\sqrt[d-1]{n}}$. Dokonca obvody hĺbky d a veľkosti $2^{\sqrt[d]{n}}$ počítajú paritu najviac na $1/2+1/2^{\Omega(\sqrt[d]{n})}$ vstupoch (pre $d>2$). Ak by sme teda vstup zvolili náhodne, výpočet takéhoto obvodu je asi tak dobrý ako hodiť si mincou. Existujú obvody hĺbky d a veľkosti $n2^{\sqrt[d]{n}}$, ktoré počítajú paritu správne na $1/2+1/2^{\sqrt[d]{n}}$ vstupoch, takže odhad je takmer tesný.*

Ďalší smer, ktorým sa dá výsledok rozšíriť, je pozrieť sa na obvody s Mod hradlami. Tvrdenie, že $\textsc{Parity} \notin \mathsf{AC}^0$ vlastne hovorí, že ak by sme obvodom pridali $\oplus$, teda $\textsc{Mod}_2$ hradlá, tieto obvody dokážu vypočítať viac ako samotné AC^0. Prirodzená ďalšia otázka potom je, čo dokážu obvody povedzme s $\textsc{Mod}_3$ hradlami? Alebo $\textsc{Mod}_5$ hradlami? Vo všeobecnosti môžeme uvažovať $\textsc{Mod}_q$ hradlo, ktoré vráti 0, ak je súčet všetkých vstupov 0 (mod q), inak vráti 1. Definujeme triedu $\mathsf{ACC}^0(m_1,\ldots,m_k)$ ako triedu jazykov akceptovaných obvodmi s konštantnou hĺbkou, polynomiálnou veľkosťou, s hradlami $\neg$, $\vee$, $\wedge$, $\textsc{Mod}_{m_1}$, $\ldots$, $\textsc{Mod}_{m_k}$ s neobmedzeným počtom vstupov.

V tejto kapitole si ukážeme dôkaz silnejšieho tvrdenia, a síce, že paritu nevieme vypočítať, ani keď pridáme $\textsc{Mod}_3$ hradlo. A naopak, $\textsc{Mod}_3$ nedokážeme spočítať v AC^0 ani keď pridáme $\textsc{Mod}_2$ hradlo:

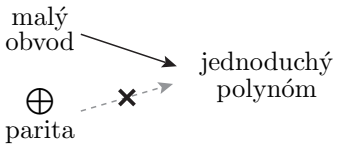
Veta 9.2 (Razborov (1987), Smolensky (1987)). $\textsc{Parity} \notin \mathsf{ACC}^0(3)$. *Vo všeobecnosti $\textsc{Mod}_p \notin \mathsf{ACC}^0(q)$ pre ľubovoľné dve prvočísla $p \neq q$.*

Dôkaz tejto vety nám zaberie celý zvyšok kapitoly, ale rozdelili sme ho do zvládnuteľných sekcii.

9.1 Aritmetizácia

Na rozdiel od Turingovych strojov, ktorým sa hlava motá kade-tade a pamäť sa im divoko prepisuje, obvody majú pomerne jednoduchú štruktúru (DAG), ktorá priam nabáda vyskúšať nasledujúcu stratégiu: Funkcie by sme chceli rozdeliť zhruba na „jednoduché“ a „ťažké“, pričom by sme dokázali (indukciou), že AND, OR, NOT jednoduchých funkcií je stále ešte „jednoduchá“ funkcia, ale parita už je zložitá.

Ostáva len vymyslieť, ako zvoliť takúto mieru zložitosti. Tu nám pomôže technika zvaná *aritmetizácia* (technika je finta, ktorú vieme použiť viac ako len raz): z obvodu na vstupe vyrobíme polynóm. Pokúsime sa dokázať zhruba toto:



1. Pre každý malý AC^0 obvod existuje „jednoduchý“ polynóm, zatiaľčo
2. pre paritu neexistuje „jednoduchý“ polynóm.

Z toho vyplýva, že parita nemá malý AC^0 obvod.

Pojem „jednoduchý“ musíme, samozrejme, ešte upresniť, ale zatiaľ si ukážme, čo majú vôbec polynómy spoločné s obvodmi.

Vezmime si napríklad obdvod, ktorý počíta $\textsc{Mod}_3(\neg x, y \wedge z, w) \vee w$. Tvrdíme, že k nemu vieme vyrobiť ekvivalentný polynóm $p(x, y, z, w)$ nad $\mathbb{Z}_3$, pričom nám stačí, ak sa výsledky budú zhodovať na vstupoch $x, y, z, w \in \{0, 1\}$. Ako na to?

- vstupnej hodnote x zodpovedá premenná x
- hradlu $\neg x$ zodpovedá výraz $1 - x$
- $x \wedge y \wedge z$ zodpovedá výraz xyz
- $x \vee y \vee z$ vieme „implementovať“ pomocou deMorganovho zákona: $1 - (1 - x)(1 - y)(1 - z)$
- $\textsc{Mod}_3(x, y, z)$ skoro zodpovedá $x + y + z$, až na drobný detail – $\textsc{Mod}_3$ vracia 0/1, zatiaľčo polynóm $x + y + z$ môže vrátiť 2; čo s tým? uvedomme si, že $2 \equiv -1 \pmod 3$, takže stačí umocniť na druhú: $(x + y + z)^2$

K obvodu $\textsc{Mod}_3(\neg x, y \wedge z, w) \vee w$ teda vieme zostrojiť ekvivalentný polynóm $1 - [1 - ((1 - x) + yz + w)^2](1 - w)$ piateho stupňa.

Prirodzená miera zložitosti polynómov je ich stupeň, mohli by sme sa teda pokúsiť dokázať, že

1. Pre každý malý AC^0 obvod existuje polynóm malého stupňa, zatiaľčo
2. pre paritu neexistuje polynóm malého stupňa.

Zatiaľ sme však videli len to, ako z obvodu vytvoriť zložitý polynóm veľkého stupňa. A vyzerá to, že oveľa lepšie sa to ani nedá. Každý AND/OR n vstupov zodpovedal súčinu n premenných, teda polynómu stupňa n, čo je príliš veľa.

9.2 Aproximácia polynómom malého stupňa

Potrebujeme teda ďalšiu fintu – a tou je aproximácia. Dokážeme, že

1. Pre každý AC^0 obvod existuje polynóm malého stupňa, ktorý ho aproximuje s exponenciálne malou chybou zatiaľčo

2. pre paritu neexistuje taký polynóm; presnejšie: ľubovoľný polynóm malého stupňa aproximuje paritu s konštantne veľkou chybou.

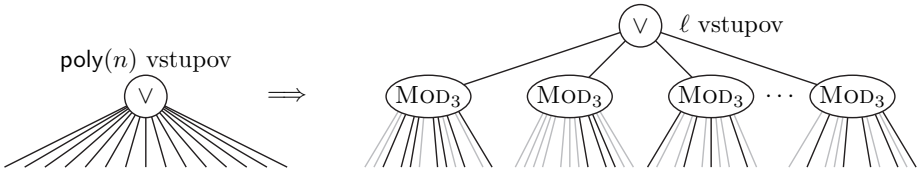
Ukážeme si, že zatiaľčo na presný výpočet AND a OR sme potrebovali polynóm n-tého stupňa, na približný výpočet stačí polynóm stupňa[1] $\sqrt{n}$.

Ako aproximovať $\mathrm{OR}(x_1, x_2, \ldots, x_n)$? OR je nulový práve vtedy, keď sú všetky vstupy nula, t.j., keď je súčet (v celých číslach) $\sum_i x_i = 0$. Dobrá správa je, že $\sum_i x_i$ je polynóm dokonca stupňa 1 – avšak my pracujeme s polynómami nad $\mathbb{Z}_3$, takže dostaneme iba súčet modulo 3. Tu nám pomôžu pravdepodobnostné algoritmy: vyberme si náhodnú podmnožinu vstupov a spočítajme ich súčet (modulo 3) – ak je vstup nulový, súčet bude vždy nula, ale ak je nenulový, s malou pravdepodobnosťou bude rôzny od nuly. Takto vieme odlíšiť nulový a nenulový vstup pomocou polynómu malého stupňa.

Ešte lepšie ako náhodnú podmnožinu môžeme uvažovať vážený súčet $\sum_i a_i x_i$, kde $a_i \in_R \{0, 1, 2\}$ sú náhodné koeficienty (náhodná podmnožina je vlastne súčet, kde koeficienty sú len 0 alebo 1). Mimochodom, $\sum_i a_i x_i$ je skalárny súčin vektorov $\vec{x}$ a $\vec{a}$. Skalárny súčin dvoch vektorov je nulový, keď sú na seba kolmé, alebo je aspoň jeden z nich nulový. Geometricky si teda tento test môžeme predstaviť nasledovne: Ak chceme zistiť, či je vektor $\vec{x}$ nulový, zvolíme si náhodný vektor $\vec{a}$ a zistíme, či sú na seba kolmé. A aká je pravdepodobnosť, že ak $\vec{x} \neq \vec{0}$, tak náhodný vektor bude na $\vec{x}$ kolmý? To, samozrejme, záleží na tom, s akým priestorom pracujeme. Ak by sme napríklad uvažovali v 3D Euklidovskom priestore $\mathbb{R}^3$, tak vektory kolmé na $\vec{x}$ tvoria rovinu (teda 2D podpriestor). Všetky ostatné vektory nie sú kolmé, tzn. ak by sme v 3D zvolili náhodný vektor $\vec{a}$, tak $\Pr_{\vec{a} \in_R \mathbb{R}^3}[\vec{x} \perp \vec{a}] = 0$ pre $\vec{x} \neq \vec{0}$. Toto je dobrá vizuálna predstava, avšak my robíme s priestorom $\mathbb{Z}_3^n$. Tvrdíme, že v ňom je $\Pr_{\vec{a} \in_R \mathbb{Z}_3^n}[\vec{x} \perp \vec{a}] = 1/3$. Jeden spôsob, ako to nahliadnuť je, že množina vektorov kolmých na $\vec{x}$ tvorí (tak ako v 3D) podpriestor $\vec{x}^\perp$ dimenzie $n-1$ – podpriestor izomorfný so $\mathbb{Z}_3^{n-1}$. Teda 3^{n-1} vektorov z 3^n je kolmých na $\vec{x} \neq \vec{0}$. Jednoduchý priamy dôkaz je, že nech $x_i \neq 0$; pre každý vektor $\vec{a}$ kolmý na $\vec{x}$ je $\vec{a} \cdot \vec{x} = 0$, ale ak zmeníme súradnicu x_i, súčet sa zmení o 1, resp. 2, t.j. pre každý kolmý vektor máme presne 2 ďalšie, ktoré nie sú kolmé.

Pravdepodobnosť 1/3, že sa pomýlime, je veľká, ale ak tento test zopárkrát zopakujeme ℓ-krát, pravdepodobnosť klesne exponenciálne na $1/3^\ell$. Podľa priemerovacieho princípu, ak pre každé x je pravdepodobnosť náhodne vybratých $\vec{a}$-čok rovná $1/3^\ell$, tak existuje aj konkrétna voľba $\vec{a}$-čok taká, že pre náhodné $\vec{x}$

[1] Samozrejme, čím presnejšie chceme hradlá aproximovať, tým zložitejšie polynómy potrebujeme; robíme trade-off medzi presnosťou a stupňom polynómu a $\sqrt{n}$ je vhodný kompromis.



Obr. 9.1: Obvod s neobmedzenými OR-mi aproximujeme tak, že spočítame náhodné súčty modulo 3. Platí totiž, že ak je $\sum_i a_i x_i \bmod 3 \neq 0$, potom $\bigvee_i x_i = 1$. Takto spočítame ℓ náhodných súčtov a ak je aspoň jeden rôzny od nuly, OR je 1. Pravdepodobnosť chyby je len $1/3^\ell$ a neobmedzený OR sme nahradili OR-om s ℓ vstupmi.

je pravdepodobnosť $\leq 1/3^\ell$:

$$\begin{aligned}
\forall \vec{x} : \Pr_{\vec{a_1},\ldots,\vec{a_\ell}}[\forall i : \vec{x} \perp \vec{a_i}] &= 1/3^\ell \\
&\Longrightarrow \Pr_{\vec{x},\vec{a_1},\ldots,\vec{a_\ell}}[\forall i : \vec{x} \perp \vec{a_i}] \leq 1/3^\ell \\
&\Longrightarrow \exists \vec{a_1},\ldots,\vec{a_\ell} : \Pr_{\vec{x}}[\forall i : \vec{x} \perp \vec{a_i}] \leq 1/3^\ell
\end{aligned}$$

Inými slovami, všetky testy zlyhajú len na $1/3^\ell$-tine vstupov.

Týchto ℓ skalárnych súčinov $\vec{x}\cdot\vec{a}_i$ umocníme na druhú (aby sme ± 1 zobrazili na 1) a výsledky spojíme naivným OR-om. Výsledný polynóm bude mať stupeň 2ℓ. AND riešime pomocou deMorganovho zákona cez OR a NOT.

Indukciou môžeme dokázať, že pre hradlo na úrovni h vieme zostrojiť polynóm stupňa $(2\ell)^h$, ktorý ho dobre aproximuje: Pre vstupné hradlá a NOT tvrdenie platí, inak sú vstupy hradla aspoň o úroveň nižšie, teda z indukčného predpokladu im prislúcha polynóm stupňa $\leq (2\ell)^{h-1}$. Konštrukcia pre MOD_3 zvýši stupeň na dvojnásobok, konštrukcia pre AND/OR stupeň zvýši 2ℓ-krát.

Tým sme dokázali, že

> pre každý obvod veľkosti S hĺbky d existuje polynóm stupňa $(2\ell)^d$, ktorý ho aproximuje s chybou $\leq S/3^\ell$.

Ak zvolíme hodnotu $2\ell = n^{1/2d}$, dostaneme polynóm stupňa $\sqrt{n}$, ktorý aproximuje s chybou $\leq S/3^{n^{1/2d}/2} \leq S/3^{n^\varepsilon}$, zatiaľčo

> ľubovoľný polynóm stupňa $\sqrt{n}$ aproximuje paritu s chybou aspoň $1/50$ (ostáva dokázať).

Čiže ak by existoval obvod konštantnej hĺbky pre paritu, potom by existoval aj aproximačný polynóm s chybou $S/3^{n^\varepsilon}$. Avšak parita sa dá aproximovať iba s chybou aspoň $1/50$. Z toho vyplýva, že $S/3^{n^\varepsilon} \geq 1/50$ a teda $S \geq 3^{n^\varepsilon}/50$.

9.3 Parita sa nedá aproximovať polynómom malého stupňa

Poďme teraz dokázať druhú časť tvrdenia: polynóm f stupňa $\sqrt{n}$ aproximuje paritu s chybou aspoň $1/50$. Označme $G' \subseteq \{0,1\}^n$ množinu vstupov, kde sa $f(\vec{x})$ a $\bigoplus(\vec{x})$ zhodujú – chceme dokázať, že G' je malá.

Tradične ako vstupy obvodu uvažujeme bity `0/1`; pri analýze booleovských obvodov (pozri napríklad O'Donnell (2014)) je však z matematického hľadiska často výhodnejšie pracovať s „bitmi“ $+1/-1$. Dosiahneme to substitúciou $y_i = 1 + x_i \pmod 3$. Tá zobrazí

$$\begin{aligned} f(\vec{x}) &\to g(\vec{y}) \quad \text{stále stupňa } \sqrt{n}\ (!) \\ 0 &\to +1, \\ 1 &\to -1, \\ b \in \{0,1\} &\to (-1)^b \in \{-1,+1\} \\ a \oplus b &\to (-1)^{a\oplus b} = (-1)^a \cdot (-1)^b \end{aligned}$$

Parita počtu jednotiek v $\vec{x}$ sa zmení na paritu počtu mínus-jednotiek v $\vec{y}$, čo je vlastne to isté ako súčin:

$$\begin{aligned} \text{parita } \bigoplus_i x_i &\to \text{súčin } \prod_i y_i \\ G' \subseteq \{0,1\}^n &\to G \subseteq \{-1,+1\}^n \end{aligned}$$

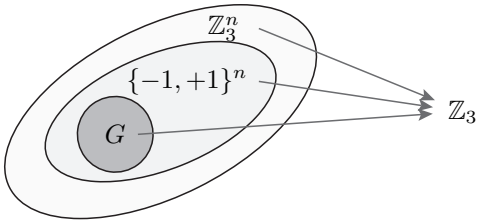
(ak je počet -1 párny, výsledok je 1, ak je nepárny, výsledok je -1). Množina $G' \subseteq \{0,1\}^n$ sa zobrazí na množinu $G \subseteq \{-1,+1\}^n$ vstupov, kde sa $g(\vec{y})$ a $\prod_i y_i$ zhodujú.

Tu si všimnime jednu zvláštu vec: stupeň polynómu $\prod_i y_i$ je až n a my ho chceme aproximovať polynómom $g(\vec{y})$ stupňa iba $\sqrt{n}$? Ukážeme, že sa to veľmi dobre nedá (množina $|G|$, kde sa zhodujú, musí byť malá).

Uvažujme funkcie $s : G \to \{-1,0,+1\}$. Takýchto funkcií je zjavne $3^{|G|}$. Ukážeme, že tieto funkcie sa dajú zapísať ako polynómy malého stupňa a tých je len $\leq 3^{(49/50)\cdot 2^n}$. Z toho bude vyplývať, že $|G| \leq \frac{49}{50} \cdot 2^n$ a teda polynóm f sa mýli na aspoň $1/50$ vstupov.

Každá funkcia $\mathbb{Z}_3^n \to \mathbb{Z}_3$ sa dá zapísať ako polynóm (rozmyslite si, že naozaj). Avšak, keďže pracujeme v $\mathbb{Z}_3$, každá funkcia sa dá zapísať ako polynóm stupňa $\leq 2n$. Prečo? V $\mathbb{Z}_3$ platí, že $a^3 \equiv a$ (Malá Fermatova veta), takže mocniny vyššie ako 2 nepotrebujeme. Keďže nás však zaujímajú iba funkcie, kde vstupy sú $a \in \{-1,+1\}$, $a^2 = 1$, takže nepotrebujeme ani druhé mocniny. Každá funkcia $\{-1,+1\}^n \to \mathbb{Z}_3$ sa teda dá zapísať ako polynóm stupňa n. Nás však zaujímajú funkcie iba z $G \to \mathbb{Z}_3$ a pre vstupy $y \in G$ platí, že $\prod_i y_i = g(y)$. Ak $\prod_{i\in I} y_i$ je nejaký člen nášho polynómu, kde $|I| > n/2$, môžeme ho nahradiť

$$\prod_{i\in I} y_i = \Big(\prod_{i\in I} y_i\Big) \cdot \underbrace{\Big(\prod_{i\notin I} y_i^2\Big)}_{=1} = \underbrace{\Big(\prod_i y_i\Big)}_{=g(\vec{y})} \cdot \Big(\prod_{i\notin I} y_i\Big) = \underbrace{g(\vec{y})}_{\deg \leq \sqrt{n}} \cdot \underbrace{\prod_{i\notin I} y_i}_{\leq n/2},$$



Obr. 9.2: Každá funkcia $\mathbb{Z}_3^n \to \mathbb{Z}_3$ sa dá reprezentovať polynómom stupňa $2n$ (tretie mocniny netreba, keďže $a^3 \equiv a \pmod 3$). Funkcie $\{-1,+1\}^n \to \mathbb{Z}_3$ sa dajú vyjadriť polynómom stupňa n (druhé mocniny netreba, pretože $a^2 = 1$). Ukážeme, že funkcie $G \to \mathbb{Z}_3$ sa dajú zapísať ako polynóm stupňa $n/2 + \sqrt{n}$ (pri tom využijeme, že G je oblasť, na ktorej platí $\prod_i y_i = g(y)$). Ukážeme, že takýchto polynómov nie je veľa a preto G je najviac 98% z množiny $\{-1,+1\}^n$. Inými slovami, polynóm g sa mýli aspoň na 1/50 vstupov.

členom, ktorý je stupňa $n/2+\sqrt{n}$. Takže každá funkcia z $G \to \mathbb{Z}_3$ sa dá vyjadriť ako polynóm stupňa $n/2+\sqrt{n}$, kde každá premenná je umocnená na 0 alebo 1.

Koľko je takýchto polynómov? Každý člen má tvar

$$\alpha \cdot y_1^{\beta_1} \cdot y_2^{\beta_2} \cdots y_n^{\beta_n},$$

kde $\beta_i \in \{0,1\}$, pričom stupeň je $\sum_i \beta_i \le n/2 + \sqrt{n}$. Všetkých možností, ako zvoliť bety je

$$\sum_{i \le n/2+\sqrt{n}} \binom{n}{i} < \frac{49}{50} \cdot 2^n$$

(pre dostatočne veľké n, pozri obrázok 9.3). Pre každý takýto monomiál môžeme zvoliť $\alpha \in \{-1,0,+1\}$, teda celkovo máme $3^{(49/50)\cdot 2^n}$ možností.

Zhrnutie: Funkcií z $G \to \mathbb{Z}_3$ je málo, preto samotná G musí byť malá a preto polynóm stupňa $\sqrt{n}$, s ktorým sme začali sa nezhoduje s paritou na aspoň 1/50-tine vstupov. □

Úlohy

- Dokážte, že AC^0 obvody nedokážu spočítať súčin dvoch n-bitových čísel. (Hint: Ukážte, že keby áno, potom by sa dala AC^0 obvodmi spočítať aj parita.)

- Dokážte, že problém MAJORITY – zistiť, či je na vstupe viac jednotiek ako núl – nepatrí do AC^0.

- Dá sa tranzitivny uzáver grafu vypočítať v AC^0?

- Dá sa n čísiel zotriediť v AC^0?

Literatúra

Furst, Merrick, James B Saxe, a Michael Sipser. 1984. "Parity, circuits, and the polynomial-time hierarchy." *Mathematical systems theory* 17(1), s. 13–27.

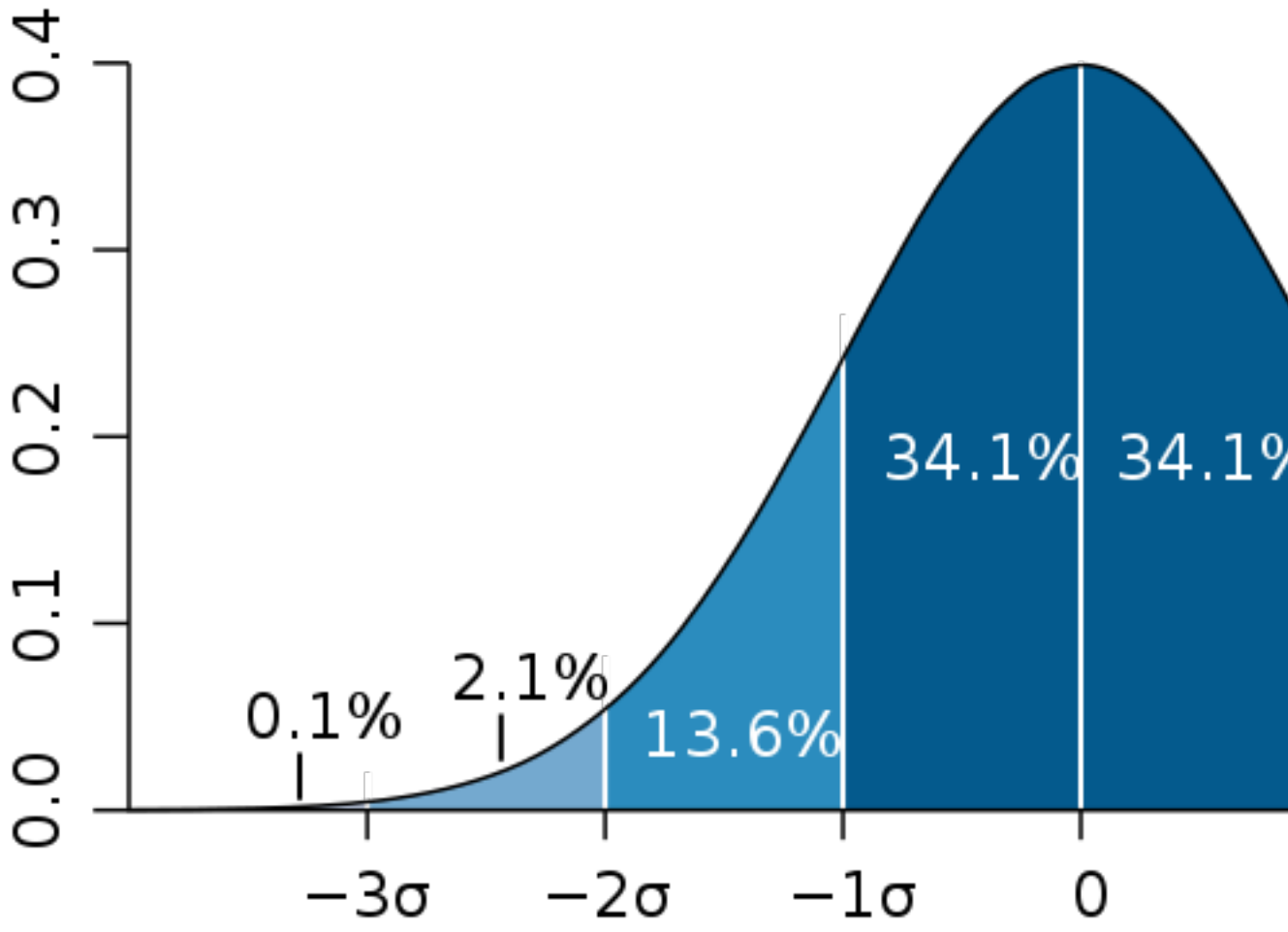
Håstad, Johan Torkel. 1987. *Computational limitations for small-depth circuits*. MIT press.

Håstad, John. 1986. "Almost optimal lower bounds for small depth circuits." In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*. s. 6–20.

O'Donnell, Ryan. 2014. *Analysis of boolean functions*. Cambridge University Press.

Razborov, Alexander A. 1987. "Lower bounds on the size of bounded depth circuits over a complete basis with logical addition." *Mathematical Notes of the Academy of Sciences of the USSR* 41(4), s. 333–338.

Smolensky, Roman. 1987. "Algebraic methods in the theory of lower bounds for Boolean circuit complexity." In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. s. 77–82.



Obr. 9.3: Súčet kombinačných čísel $\sum_{i \leq n/2+\sqrt{n}} \binom{n}{i}$ je $2^n \times \Pr_{X \sim B(n,1/2)}[X \leq n/2 + \sqrt{n}]$. Ak binomiálnu distribúciu aproximujeme normálnou distribúciou $N(\mu, \sigma^2)$, kde $\mu = n/2$ a $\sigma = \sqrt{n/2}$, plocha po 2σ zaberá menej ako 98%.

Kapitola 10

Vetviace programy

Vetviaci program (*branching program*) na vstupe $x_1 \dots x_n$ je jednoduchý výpočtový model reprezentovaný orientovaným acyklickým grafom.

Jeden vrchol je začiatočný, dva špeciálne vrcholy sú koncové (akceptuj / odmietni). S výnimkou koncových vrcholov sú vrcholy vždy označené číslom 1 až n a vychádzajú z nich 2 hrany označené 0 a 1. Výpočet je cesta v grafe, ktorá začína v začiatočnom vrchole a keď vôjde do vrcholu označeného i, pokračuje hranou označenou x_i, teda podľa i-teho bitu vstupu sa rozhodne, ktorou hranou pôjde ďalej. My budeme ďalej predpokladať, že všetky vrcholy sa dajú rozdeliť do vrstiev tak, že všetky hrany z jednej vrstvy smerujú do tej nasledovnej; prvá vrstva obsahuje iba začiatočný vrchol a posledná dva koncové vrcholy.

Definovali sme jednoduchú verziu deterministického vetviaceho programu nad binárnou abecedou. Ak by sme chceli, mohli by sme uvažovať hrany označené písmenami väčšej abecedy, alebo nedeterministický model, kde z jedného vrcholu môže vychádzať viacero hrán označených tým istým znakom.

Vetviaci program funguje len pre konkrétnu dĺžku vstupu n. Ak chceme vedieť riešiť problémy s ľubovoľnou dĺžkou vstupu, uvažujeme, podobne ako pri obvodoch, že máme celú triedu vetviacich programov – pre každú dĺžku vstupu jeden. A podobne ako pri obvodoch môžeme uvažovať ľubovoľnú triedu alebo uniformnú, kde n-tý vetviaci program vieme efektívne zostrojiť (resp. môžeme uvažovať rôzne stupne uniformnosti).

Tak ako pri TS meriame ich čas a pamäť, pri obvodoch veľkosť a hĺbku, vhodné miery zložitosti pre vetviace programy sú

1. *veľkosť* – počet vrcholov
2. *dĺžka* – počet vrstiev
3. *šírka* – počet vrcholov v jednej vrstve

Akú silu majú vetviace programy? Záleží, či máme na mysli uniformnú alebo neuniformnú verziu. Neuniformné vetviace programy polynomiálnej veľkosti sa dajú simulovať polynomiálnymi obvodmi, teda $\mathsf{PBP} \subseteq \mathsf{P/poly}$. Nie je však jasné,

či platí aj opačná inklúzia. Ak si vezmeme definíciu $\mathsf{P/poly}$ cez TS s radou, môžeme si všimnúť, že vetviace programy sa dajú simulovať s malou (logaritmickou) pamäťou (a polynomiálnou radou): $\mathsf{PBP} \subseteq \mathsf{L/poly}$. Dá sa dokázať, že v tomto prípade platí rovnosť: $\mathsf{PBP} = \mathsf{L/poly}$ (Pudlák a Žák, 1983). Pre logspace-uniformné programy platí $\mathsf{unif\text{-}PBP} = \mathsf{L}$.

Nás však bude v tejto kapitole zaujímať, akú silu majú vetviace programy *konečnej šírky*. Skúste si rozmyslieť, že:

1. Všetky regulárne jazyky sa dajú vypočítať programmi konštantnej šírky.
2. Programmi konštantnej šírky však vieme spočítať aj jazyk palindrómov či $\{a^n b^n c^n \mid n \in \mathbb{N}\}$.
3. AND, OR, XOR (parita) sa dá spočítať programom šírky 2 dĺžky $O(n)$.
4. Každá funkcia sa dá vypočítať vetviacim programom exponenciálnej dĺžky šírky 3.
5. Aký najkratší program konštantnej šírky dokáže spočítať majoritu?

Veta 10.1 (Barrington (1989)). *Pre každý obvod hĺbky d existuje ekvivalentný vetviaci program šírky 5 dĺžky $\ell = 4^d$. To znamená pre $d = O(\log n)$ je $\ell = \mathsf{poly}(n)$ a teda $\mathsf{NC}^1 = 5\text{-}\mathsf{PBP} = k\text{-}\mathsf{PBP}$. V oboch prípadoch máme na mysli neuniformné triedy, hoci podobná veta platí aj pre uniformné triedy.*

Prečo práve 5? Čo je na päťke také špeciálne?? Odpoveď prichádza z nečakaného smeru: z teórie grúp. Súvisí totiž s tým, že S_5 – grupa permutácií nie je riešiteľná.

Dôkaz, že vetviace programy konštantnej šírky vieme počítať v NC^1, nechávame ako úlohu pre čitateľa. V nasledujúcej sekcii sa pozrieme na opačnú inklúziu.

10.1 Permutačné programy

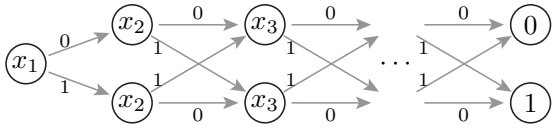
Ak sú v každej vrstve vrcholy označené rovnako (rozhodujú sa podľa toho istého bitu), potom vetviaci program môžeme zapísať ako trojicu

$$(f_1^0, f_2^0, \ldots, f_\ell^0),$$
$$(f_1^1, f_2^1, \ldots, f_\ell^1),$$
$$(k_1, k_2, \ldots, k_\ell),$$

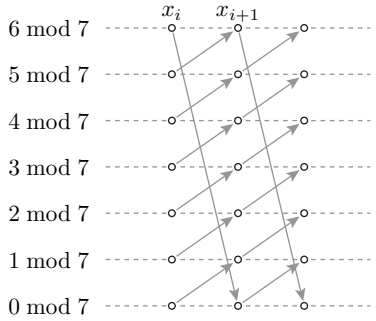
kde zobrazenie f_i^j určuje pre každý vrchol i-tej vrstvy, kam sa pohneme, ak je k_i-ty bit vstupu rovný j. Výpočet prebieha tak, že začneme v počiatočnom vrchole, v i-tom kroku sa pozrieme na k_i-ty bit; ak je $x_{k_i} = \mathtt{0}$, hýbeme sa podľa zobrazenia f_i^0, ak je to $\mathtt{1}$ ideme podľa f_i^1.

Vetu 10.1 dokonca dokážeme pre vetviace programy veľmi špeciálneho tvaru: v každej vrstve budeme mať rovnaký počet vrcholov a zobrazenia f_i^j budú *permutácie*.

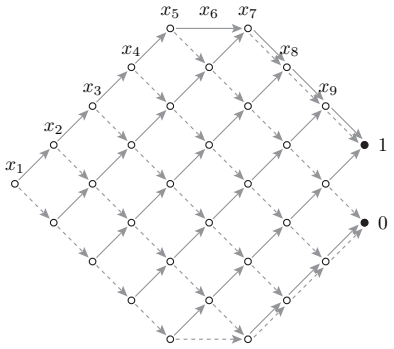
Permutačný program dĺžky ℓ je daný trojicou



(a) Vetviaci program počítajúci paritu.



(b) Zvyšok modulo m vieme vypočítať programom šírky m – každý riadok reprezentuje jednu zvyškovú triedu, ak je ďalší bit 0, ostávame v rovnakom riadku, ak je to 1, ideme o riadok mod m vyššie. Vedeli by sme zvyšok mod m vypočítať aj programom šírky $< m$ (polynomiálnej veľkosti)?



(c) Program počítajúci majoritu, teda či je na vstupe viac núl alebo jednotiek. Pozícia (i, j) zodpovedá situácii, že sme videli $\geq i$ núl a $\geq j$ jednotiek na vstupe. Takýto program má šírku zhruba $n/2$. Vedeli by sme majoritu spočítať aj programom s oveľa menšou šírkou?

Obr. 10.1: Príklady vetviacich programov. Zhodou okolností testujeme v každom stĺpci len jednu premennú. Na obrázkoch dolu ideme plnou čiarou, ak je premenná = 1 a prerušovanou, ak je premenná = 0.

$$(\pi_1^0, \pi_2^0, \ldots, \pi_\ell^0),$$

$$(\pi_1^1, \pi_2^1, \ldots, \pi_\ell^1),$$

$$(k_1, k_2, \ldots, k_\ell),$$

kde $\pi_i^j \in S_m$ sú permutácie na m vrcholoch a $k_i \in [n]$.

Keď budeme vytvárať permutačné programy, budeme spájať kratšie programy do dlhších a pri tomto spájaní budeme chcieť posunúť čo najviac informácie z jedného podprogramu do ďalšieho – neoplatí sa nám preto začínať v jedinečnom vrchole a končiť v jednom z dvoch. Prvá aj posledná vrstva budeme mať m vrcholov.

Namiesto sledovania cesty z jedného vrcholu sa zase oplatí sledovať cestu z každého možného začiatku. Pozornosť budeme sústrediť nie na jednotlivé vrcholy, ale na spoje medzi nimi. V každom kroku sa pohneme podľa jednej per-

mutácie a ak chceme vedieť, kam sa dostaneme na viac krokov, stačí vhodné permutácie zložiť.

Výpočet bude prebiehať tak, že začneme s identitou, v i-tom kroku sa pozrieme na k_i-ty bit; ak je to nula, pripočítame π_i^0, inak pripočítame π_i^1. Skrátene sa teda dá povedať, že v i-tom kroku pripočítame permutáciu $\pi_i^{x_{k_i}}$. Výsledkom bude zloženie všetkých týchto permutácií od 1 po ℓ. Ostáva rozhodnúť sa, ktoré permutácie budú zodpovedať výsledku Áno a ktoré výsledku Nie.

My budeme v skutočnosti navrhovať také permutačné programy, kde na konci buď vyjde identita, vtedy je odpoveď Nie, alebo na konci vyjde nejaká permutácia $\alpha \neq \mathrm{id}$ a vtedy je odpoveď Áno. Budeme hovoriť, že program *počíta* funkciu $f : \{0,1\}^* \to \{0,1\}$, ak pre každé x platí:

$$f(x) = 0 \implies \prod_{i=1}^{\ell} \pi_i^{x_{k_i}} = 1_G$$

$$f(x) = 1 \implies \prod_{i=1}^{\ell} \pi_i^{x_{k_i}} = \alpha$$

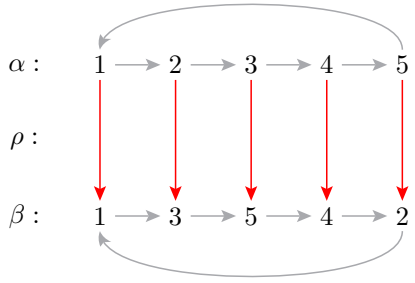
Alebo skrátene

$$\prod \pi_i^{x_{k_i}} = \alpha^{f(x)}.$$

Takéto permutačné programy vieme jednoducho implementovať ako vetviace programy šírky m: Nech x je prvok, pre ktorý $\alpha(x) \neq x$. Namiesto prvej vrstvy necháme iba prvok x a v poslednej vrstve bude $\alpha(x)$ akceptujúci a x odmietací vrchol.

Lema 10.1. *Nech α, β sú 5-cykly. Potom f je α-vypočítateľná programom dĺžky ℓ práve vtedy, keď je β-vypočítateľná programom dĺžky ℓ.*

■ **Dôkaz.** Pre ľubovoľné dva cykly $\alpha = (a_1, \ldots, a_n)$ a $\beta = (b_1, \ldots, b_n)$ existuje permutácia ρ taká, že $\alpha = \rho \circ \beta \circ \rho^{-1}$ a naopak $\beta = \rho^{-1} \circ \alpha \circ \rho$ (hovoríme, že α a β sú navzájom konjugované). Hľadaná permutácia je $\rho(a_k) = b_k$, čo ľahko overíme výpočtom: $\rho^{-1}(\beta(\rho(a_k))) = \rho^{-1}(\beta(b_k)) = \rho^{-1}(b_{k+1}) = a_{k+1} = \alpha(a_k))$.



Stačí potom zobrať pôvodný program a prenásobiť prvé prvky ρ^{-1} zľava a posledné ρ sprava, t.j. namiesto π_1^j sa použije $\rho^{-1}\pi_1^j$ a namiesto π_ℓ^j sa použije $\pi_\ell^j \rho$. Ak pôvodný program počítal hodnotu π, nový program má rovnakú dĺžku,

ale výsledok bude $\rho^{-1} \circ \pi \circ \rho$, teda α namiesto β a v prípade identity sa hodnota nezmení: $\rho^{-1} \circ \mathrm{id} \circ \rho = \rho^{-1} \circ \rho = \mathrm{id}$. □

Lema 10.2. *Ak f je α-vypočítateľná programom dĺžky ℓ, tak aj $\neg f$ je.*

■ **Dôkaz.** Podľa predchádzajúcej lemy je f α^{-1}-vypočítateľná. Stačí posledné prvky v tomto programe vynásobiť α. Ak bol pôvodný výsledok α^{-1}, teraz bude id a naopak, ak bol predtým id, teraz bude α. Program teda α-počíta $\neg f$.
□

Lema 10.3. *Ak f je α-vypočítateľná a g je β-vypočítateľná programom dĺžky ℓ, tak $f \wedge g$ je $(\alpha\beta\alpha^{-1}\beta^{-1})$-vypočítateľná programom dĺžky 4ℓ.*

■ **Dôkaz.** Program pre $f \wedge g$ najskôr α-vypočíta f, potom β-vypočíta g, potom α^{-1}-vypočíta f a nakoniec β^{-1}-vypočíta g. Ak $f(x) = g(x) = 1$, výsledok bude $\alpha \circ \beta \circ \alpha^{-1} \circ \beta^{-1}$. Ak je však aspoň jedna hodnota 0, príslušné členy zmiznú a zvyšné sa vykrátia: napr. ak $f(x) = 0$, potom výpočet $f(x)$ vráti identitu a výsledok bude $\mathrm{id} \circ \beta \circ \mathrm{id}^{-1} \circ \beta^{-1} = \beta \circ \beta^{-1} = \mathrm{id}$. □

Dôkaz vety je založený na tom, že existujú cykly α a β, pre ktoré je $\alpha \circ \beta \circ \alpha^{-1} \circ \beta^{-1}$ tiež cyklus (a teda *nie je* identita). Na to potrebujeme permutácie aspoň piatich prvkov.

Konkrétne, nech napríklad

$$\begin{aligned} \alpha &= 1 \to 2 \to 3 \to 4 \to 5 \to 1 \\ \beta &= 1 \to 3 \to 5 \to 4 \to 2 \to 1 \\ \alpha\beta\alpha^{-1}\beta^{-1} &= 1 \to 3 \to 2 \to 5 \to 4 \to 1 \end{aligned}$$

■ **Dôkaz Barringtonovej vety.** Indukciou od hĺbky s využitím predchádzajúcich liem: Pri $f \wedge g$ programy pre vstupy vždy upravíme tak, aby $\alpha = (1,2,3,4,5)$-počítali f a $\beta = (1,3,5,4,2)$-počítali g, výsledný program bude $\alpha\beta\alpha^{-1}\beta^{-1} = (1,4,3,5,2)$-počítať $f \wedge g$. Hradlo $f \vee g$ počítame pomocou de-Morganovho pravidla $f \vee g = \neg(\neg f \wedge \neg g)$. Pre hradlo vo výške d majú vstupy výšku aspoň o 1 menšiu a teda z indukčného predpokladu majú programy pre vstupy dĺžku $\leq 4^{d-1}$; podľa liem bude program pre $\wedge$ alebo $\vee$ 4-krát dlhší.

Takto získame permutačný program dĺžky 4^d, ktorý triviálne implementujeme vetviacim programom šírky 5. □

Malá odbočka do teórie grúp: Permutačné programy môžeme zovšeobecniť na grupové programy, či dokonca programy na monoidoch. Dôkaz Barringtonovej vety využíva súčin $[a,b] = aba^{-1}b^{-1}$, ktorý sa v teórii grúp volá *komutátor*. Platí totiž, že $[a,b] = 1$ práve vtedy, keď je násobenie a a b komutatívne ($ab = ba$).

Prečo veta platí pre programy šírky 5 a nie menej? Dôkaz využíva štruktúru grúp, takzvanú *neriešiteľnosť*.[1] Definujme komutátorovú podgrupa grupy G ako podgrupu G' generovanú všetkými komutátormi v G, t.j. $G' = \langle\{[a,b] \mid a,b \in G\}\rangle$ a označme $G^{(0)} = G$ a $G^{(k+1)} = (G^{(k)})'$. Postupnosť $G \supseteq G' \supseteq G'' \supseteq \cdots \supseteq G^{(k)}$ voláme *komutátorový rad* grupy G. Hovoríme, že grupa je riešiteľná, ak jej komutátorový rad končí triviálnou grupou 1.

Napríklad S_4, grupa permutácií 4-och prvkov je riešiteľná – má rad

$$S_4 > A_4 > V_4 > 1$$

kde A_4 je alternujúca grupa (párne permutácie) a V_4 je tzv. Kleinova 4-grupa $\{\mathrm{id}, (1,2)(3,4), (1,3)(2,4), (1,4)(2,3)\}$. S takouto grupou by náš dôkaz nefungoval, pretože komutátor ľubovoľných dvoch permutácií spadne do A_4 a komutátor ľubovoľných párnych permutácií spadne do V_4 a V_4 už je komutatívna. Naproti tomu $S_5 > A_5$, ale $A_5' = A_5$. A_5 je najmenšia grupa, ktorá nie je riešiteľná.

Barrington a Therien (1988) dokázali, že 4-$\mathsf{PBP} \subseteq \mathsf{ACC}^0$, takže šírku aspoň 5 treba. Presnejšie, výpočty grupových programov polynomiálnej veľkosti na riešiteľných grupách zodpovedajú presne triede ACC^0. Trieda AC^0 sa zase dá charakterizovať polynomiálne veľkými programami na aperiodických monoidoch (takých, že pre každý prvok m existuje k také, že $m^k = m^{k+1}$).

10.2 Dôsledky

Dôsledok 10.1. *Existuje regulárny jazyk, ktorý je NC^1-úplný pri $\leq_m^{\mathsf{AC}^0}$-redukcií, tzn. každý problém v NC^1 sa dá AC^0 obvodom redukovať na regulárny jazyk.*

■ **Dôkaz.** Pre ľubovoľnú konečnú grupu G a prvok $g \in G$ je jazyk $L_{G,g} = \{w_1 \cdots w_n \mid \prod_i w_i = g\}$ regulárny. Tvrdíme, že $L_{S_5,\alpha}$ je NC^1-úplný.

Nech $L \in \mathsf{NC}^1$. Barringtonova veta ukazuje, ako pre NC^1-obvod vyrobiť ekvivalentný permutačný program; AC^0-obvod na vstupe $x_1 \cdots x_n$ zostrojí slovo $w = \pi_1^{x_{k_1}} \cdots \pi_\ell^{x_{k_\ell}}$, teda pre každé i podľa vstupu zvolí π_i^0 alebo π_i^1. Slovo x patrí do jazyka L práve vtedy, keď $w \in L_{S_5,\alpha}$. To znamená $L \leq_m^{\mathsf{AC}^0} L_{S_5,\alpha}$. □

Predstavme si počítač, ktorý má okrem operačnej pamäte ešte jedno počítadlo, malé bezpečné úložisko (hard-disk) a jeden malý háčik: výpočet prebieha vo fázach, pričom každý deň počítač začína s prázdnou pamäťou; spraví polynomiálne veľa krokov a tie najdôležitejšie informácie si nahrá na úložisko. Na

[1] Tento pojem pochádza z Galoisovej teórie a bol motivovaný snahou pochopiť riešiteľnosť rovníc piateho stupňa. Tak ako sa dajú riešiť kvadratické rovnice vzorčekom $x = (-b \pm \sqrt{D})/(2a)$, kde $D = b^2 - 4ac$, dajú sa riešiť aj kubické rovnice: $x = \sqrt[3]{-q/2 + \sqrt{q^2/4 + p^3/27}} + \sqrt[3]{-q/2 - \sqrt{q^2/4 + p^3/27}} - b/3a$, kde $p = (3ac - b^2)/3a^2$, $q = (2b^3 - 9abc + 27a^2d)/27a^3$ a kvartické rovnice (pozri `https://en.wikipedia.org/wiki/Quartic_function`, vzorček sa nezmestí na okraj tejto knihy). Pre kvintické rovnice (rovnice piateho stupňa tvaru $ax^5 + bx^4 + cx^3 + dx^2 + ex + f = 0$) neexistuje takýto vzorček zložený z koeficientov, konštánt, $+$, $-$, $\times$, / a odmnocnín $\sqrt[n]{\cdot}$. Tento výsledok je známy ako Abelova-Ruffiniho veta.

večer sa počítač vypína a všetky dáta v RAMke sa vymažú (ostanú len dáta na hard-disku). Na druhý deň sa počítadlo zvýši o 1 a výpočet môže pokračovať. Program nemôže prepisovať ani vstup ani počítadlo a tak jediná informácia, ktorú si môže preniesť z jedného dňa na druhý, je to, čo si zapíše na bezpečné úložisko. Výpočet končí, keď počítadlo pretečie (ak máme t-bitové počítadlo, tak po 2^t krokoch) a výsledok záleží od hodnoty na úložisku.

Tak napríklad klasické Turingove stroje s polynomiálnou pamäťou sú ekvivalentné počítačom s polynomiálne veľkým úložiskom a polynomiálne veľkým počítadlom (tým pádom si môžu celú konfiguráciu zapísať na úložisko a výpočet môže byť exponenciálne dlhý).

Uvažujme však oveľa oklieštenejší model:

- každá fáza je v logaritmickej pamäti
- bezpečné úložisko má 3 bity (slovom: tri bity!)
- počítadlo má polynomiálny počet bitov.

Problémy, ktoré sa dajú riešiť v takomto modeli voláme *logspace serializovateľné*. Otázka: aké problémy sú logspace serializovateľné?

Pripomeňme, že podľa vety o pamäťovej hierarchii vieme, že $\mathsf{L} \subsetneq \mathsf{PSPACE}$, takže napríklad QBF dokázateľne potrebuje viac ako len logaritmický priestor. Navyše uložiť si 3 bity je zjavne takmer nanič. Napriek tomu, prekvapujúco:

Veta 10.2 (Cai a Furst (1991)). *Všetky problémy v* PSPACE *sú logspace serializovateľné.*

■ **Náznak dôkazu.** PSPACE problémy sú riešiteľné exponenciálne veľkými obvodmi polynomiálnej hĺbky a z Barringtonovej vety vyplýva, že sú riešiteľné vetviacimi/permutačnými programmi exponenciálnej dĺžky a šírky 5. Dá sa navyše ukázať, že i-ty bit a i-tu permutáciu dokážeme vypočítať v logaritmickom priestore. Výpočet bude prebiehať nasledovne: simulujeme permutačný program, pričom v úložisku si pamätáme hodnotu vrcholu $\{1, 2, \ldots, 5\}$, v ktorom sme. Začíname v 1. V i-tej fáze (podľa počítadla) zistíme, na ktorý bit sa pozrieť a jeho hodnotu, následne podľa toho skonštruujeme i-tu permutáciu a zistíme hodnotu nového vrcholu, ktorú zapíšeme na úložisko. Na konci sme buď v 1 (výsledok bola identita, odpoveď je NIE), alebo inde (odpoveď je ÁNO). □

Algebraické obvody sú zovšeobecnenie boolovských obvodov pre ľubovoľný okruh $(K, +, \times, 0, 1)$. Namiesto AND, OR a NOT hradiel máme hradlá $+$, $\times$ a na vstupe sú premenné x_i alebo špeciálne konštantné hradlá $c \in K$.

Veta 10.3 (Ben-Or a Cleve (1992)). *Majme funkciu danú algebraickým obvodom hĺbky d. Tá istá funkcia sa dá vypočítať súčinom 4^d matíc 3×3. Funkcie vypočítateľné algebraickými* NC^1 *obvodmi sú presne tie, ktoré sa dajú spočítať súčinom polynomiálneho počtu matíc 3×3.*

■ **Dôkaz.** Rekurzívna konštrukcia: predpokladajme, že pre obvody $f = f(x_1, \ldots, x_n)$ a $g = g(x_1, \ldots, x_n)$ skonštruujeme matice tvaru

$$F = \begin{pmatrix} 1 & 0 & 0 \\ f & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \text{a} \qquad G = \begin{pmatrix} 1 & 0 & 0 \\ g & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Potom

$$\begin{pmatrix} 1 & 0 & 0 \\ f+g & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = F \cdot G$$

$$\begin{pmatrix} 1 & 0 & 0 \\ f \times g & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ -f & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & g & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ f & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & -g \end{pmatrix}$$

Súčet $f + g$ dostaneme ako súčin dvoch matíc a súčin $f \times g$ ako súčin štyroch matíc podobného tvaru (konjugovaných s F a G). Napríklad

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & -g \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \cdot G \cdot \begin{pmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Tieto pomocné matice iba spermutujú, prípadne znegujú niektoré prvky.

□

Mimochodom pre okruh $K = (\mathbb{Z}_2, +, \cdot, 0, 1) = (\mathbb{Z}_2, \oplus, \wedge, 0, 1)$ dostávame Barringtonovu vetu (pomocou grupy matíc 3×3 namiesto S_5).

Uvažujme (algebraické) programy s r registrami $R_1, \ldots, R_r$ s inštrukciami

- $R_i \mathrel{+}= c$, $R_i \mathrel{-}= c$, (t.j. $R_i \leftarrow R_i \pm c$)
- $R_i \mathrel{+}= x_k$, $R_i \mathrel{-}= x_k$,
- $R_i \mathrel{+}= c \cdot R_j$, $R_i \mathrel{-}= c \cdot R_j$,
- $R_i \mathrel{+}= x_k \cdot R_j$, $R_i \mathrel{-}= x_k \cdot R_j$,

kde c je nejaká konštanta a $x_1, \ldots, x_n$ sú vstupy.

Veta 10.4 (Cleve (1991), Ben-Or a Cleve (1992)). *Každá funkcia f s algebraickým obvodom hĺbky d sa dá spočítať algebraickým programom dĺžky $O(4^d)$ s iba tromi registrami.*

■ **Dôkaz.** Indukciou budeme vytvárať programy, ktoré počítajú $R_i \pm\!= R_j \cdot g(x_1, \ldots, x_n)$ pre rôzne $i \neq j \in \{1, 2, 3\}$, pričom ostatné registre sa nezmenia. Cieľom je vytvoriť program, ktorý počíta $R_1 \mathrel{+}= R_2 \cdot f(x_1, \ldots, x_n)$.

Pre jednotlivé premenné x_k je program triviálny. Ak už máme programy pre $R_i \mathrel{+}= R_j \cdot g$ a $R_i \mathrel{+}= R_j \cdot h$, potom program pre $R_i \mathrel{+}= R_j \cdot (f + g)$ je

$$\begin{aligned} R_i &\mathrel{+}= R_j \cdot g \\ R_i &\mathrel{+}= R_j \cdot h \end{aligned}$$

a program pre $R_i \mathrel{+}= R_j \cdot (f \cdot g)$ je

$$\begin{aligned} R_i &\mathrel{-}= R_k \cdot g && // \; R_i = r_i - r_k \cdot g \\ R_k &\mathrel{+}= R_j \cdot f && // \; R_k = r_k + r_j \cdot f \\ R_i &\mathrel{+}= R_k \cdot g && // \; R_i = r_i - r_k \cdot g + (r_k + r_j \cdot f) \cdot g \\ R_k &\mathrel{-}= R_j \cdot f && // \; R_k = r_k \end{aligned}$$

(v komentároch r_i, r_j, r_k označujú pôvodnú hodnotu registrov R_i, R_j, R_k). Podobne vytvoríme programy s mínusom. □

Literatúra

Barrington, David A. 1989. "Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1." *Journal of Computer and System Sciences* 38(1), s. 150–164.

Barrington, David A Mix a Denis Therien. 1988. "Finite monoids and the fine structure of NC^1." *Journal of the ACM (JACM)* 35(4), s. 941–952.

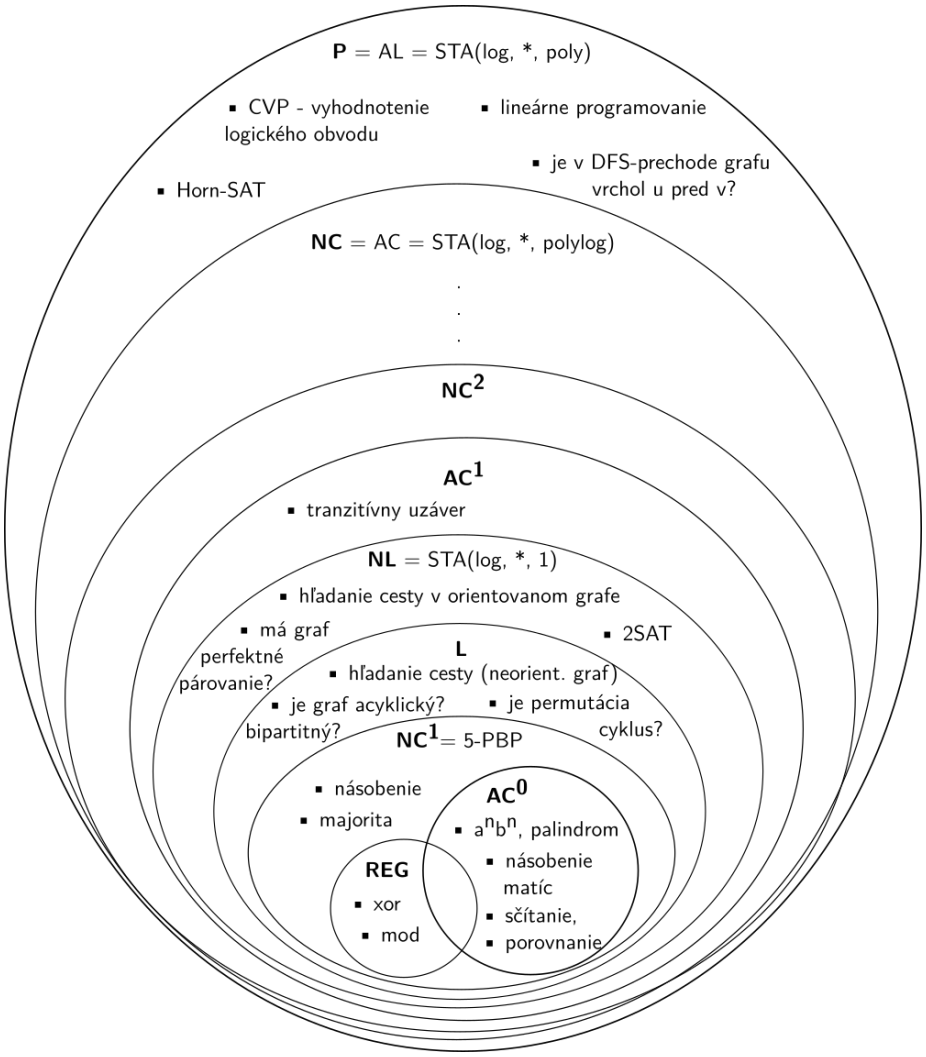
Ben-Or, Michael a Richard Cleve. 1992. "Computing algebraic formulas using a constant number of registers." *SIAM Journal on Computing* 21(1), s. 54–58.

Cai, Jin-Yi a Merrick Furst. 1991. "PSPACE survives constant-width bottlenecks." *International Journal of Foundations of Computer Science* 2(01), s. 67–76.

Cleve, Richard. 1991. "Towards optimal simulations of formulas by bounded-width programs." *Computational Complexity* 1(1), s. 91–105.

Pudlák, Pavel a Stanislav Žák. 1983. "Space complexity of computations." *Preprint Univ. of Prague* .

Zhrnutie


P = AL = STA(log, *, poly)
CVP - vyhodnotenie logického obvodu
lineárne programovanie
je v DFS-prechode grafu vrchol u pred v?
Horn-SAT
NC = AC = STA(log, *, polylog)
NC2
AC1
tranzitívny uzáver
NL = STA(log, *, 1)
hľadanie cesty v orientovanom grafe
má graf perfektné párovanie?
2SAT
L
hľadanie cesty (neorient. graf)
je graf acyklický?
je permutácia cyklus?
bipartitný?
NC1 = 5-PBP
násobenie
majorita
AC0
anbn, palindrom
násobenie matíc
sčítanie, porovnanie
REG
xor
mod

Časť IV

Ťažké hry

Úvod

Táto časť bude hravá – budeme sa zaoberať zložitosťou hier, teda otázkami ako napríklad:

- Aké ťažké je vyriešiť rôzne puzzle ako bludisko, Sudoku, či Sokoban?
- Sú behačky a skákačky ako napríklad Super Mario, alebo strielačky ako napr. Doom či Quake v nejakom zmysle ťažké?
- A aké ťažké sú hry, kde proti sebe hrajú dvaja hráči ako napríklad dáma, šach, piškvorky, Go, či Reversi?

Nebudú nás zaujímať konečné hry ako napríklad piškvorky 3×3, či dáma na šachovnici 8×8 – tie sú z hľadiska teórie zložitosti triviálne, keďže majú len konečný počet stavov a stratégií. Zložitosť je $O(1)$.

Budú nás zaujímať vhodné zovšeobecnenia týchto hier – napríklad piškvorky na ploche $n \times n$.

Kapitola 11

NP-ťažké hry

11.1 Tetris

Tetris sa tradične hrával na displayoch 20×40, my budeme uvažovať zovšeobecnenú verziu, plochu $M \times N$. Na tetrise je navyše ťažké, že nevieme dopredu, aké kocky budú nasledovať. My však budeme uvažovať hru, kde je známa postupnosť všetkých kociek, v poradí, v akom budú postupne prichádzať. Budeme uvažovať scenár, kde sa kamarát chvíľu hrá a potom nám rozohratú hru podá, že „dohraj to“. Na hracej ploche teda už nejaké kocky sú a otázka znie: dokážeme v takejto pozícii prežiť? Alebo: dajú sa vymazať všetky riadky?

Ukážeme, že takýto problém je NP-ťažký. Dôkaz bude prebiehať redukciou z problému 3-Partition: Dané sú celé čísla $a_1, \ldots, a_{3s}$. Dajú sa tieto čísla rozdeliť do s trojíc tak, aby každá trojica mala rovnaký súčet $T = \sum a_i/s$?

Napríklad

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 11$$

vieme rozdeliť do trojíc:

$$(1, 7, 11) \qquad (2, 8, 9) \qquad (3, 6, 10) \qquad (4, 5, 10)$$

Každá trojica dáva súčet 19.

Iný príklad:

$$0, 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 20$$

V tomto prípade by mala mať každá trojica súčet 30, ale číslo 5 nevieme doplniť do žiadnej takej trojice, takže odpoveď je Nie.

Problém 3-Partition je silne NP-úplný, to znamená, NP-úplný aj v prípade, že čísla na vstupe sú zakódované unárne. Dokonca môžeme predpokladať, že všetky čísla a_i sú medzi $T/4$ a $T/2$. Ukážeme, že inštancia 3-Partition sa dá „zakódovať“ v hre Tetris tak, že hra sa dá vyhrať práve vtedy, keď sa čísla dajú rozdeliť do trojíc. To znamená, ak by existoval rýchly (polynomiálny) algoritmus, ktorý zistí, či sa daná pozícia Tetrisu dá vyhrať, tak by existoval aj rýchly algoritmus pre 3-Partition a P = NP.

Ešte krátka poznámka: V tetrisovej hantírke sa jednotlivé tetrominá značia písmenami, na ktoré sa podobajú. Kocky `I`, `T`, `L` sú asi zrejmé, `O` je štvorcové tetromino, `J` je opačné `L` (tetrominá `N` a `Z` nebudeme používať).

Veta 11.1 (Breukelaar a spol. (2004)). *Hra* Tetris *je* NP*-úplná. Presnejšie: Je daný počiatočný stav hracej plochy, kde už nejaké kocky sú a je známa postupnosť všetkých kociek, v poradí, v akom budú postupne prichádzať. Otázky ako „Dá sa v tejto pozícii prežiť? Dajú sa vymazať všetky riadky?“ sú* NP*-úplné.*

■ **Dôkaz.** Pozíciu zostrojíme ako na obr. 11.1. Vyhrať sa dá práve vtedy, keď vieme jednotlivé „jamy“ presne zaplniť, potom „odomknúť zámok“ vpravo hore a vyplniť voľnú plochu vpravo, čím zmažeme všetky riadky. Postupnosť kociek bude takáto:

1. V prvej fáze najskôr pre každé a_i príde postupnosť kociek `L` (začiatok), následne a_i-krát trojica `O`, `J`, `O`, a na koniec `O`, `I`. Na obr. 11.2a vidno, ako sa dá takouto postupnosťou začať vypĺňať jama. Všimnite si, ako kocky `O`, `I` na konci uvedú jamu do „pôvodného stavu“, ktorý treba opäť začať `L`-kom.

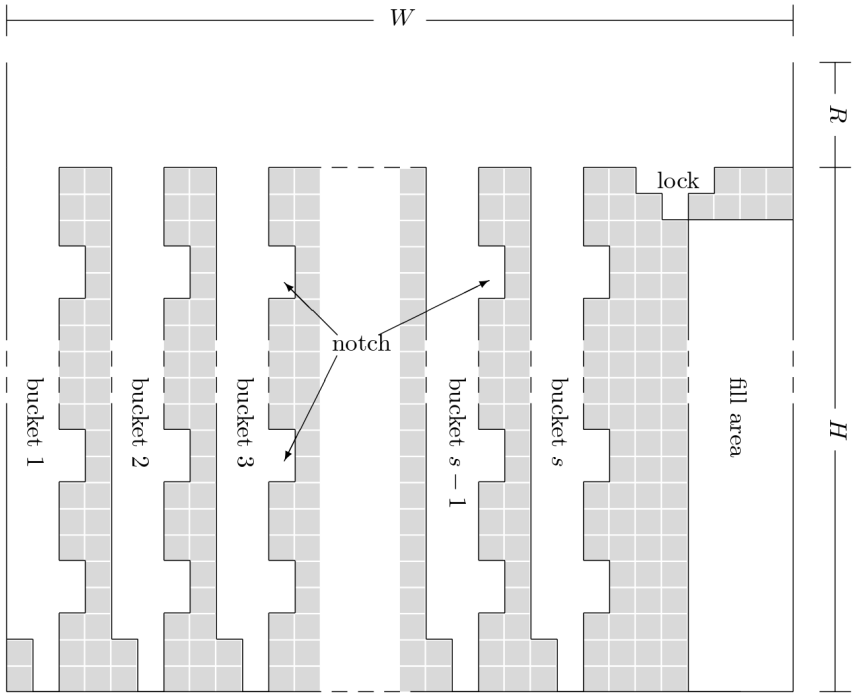
2. V druhej fáze príde $s\times$`L`, ktoré uzatvorí všetky presne vyplnené jamy ako na obr. 11.2b.

3. Nakoniec príde vykúpenie v podobe `T`, ktorým odopkneme zámok, vymažeme vrchné dva riadky, odkryjeme voľnú plochu vpravo a $5T+16$ `I`-čok, ktorými ju vyplníme a zmažeme všetky riadky.

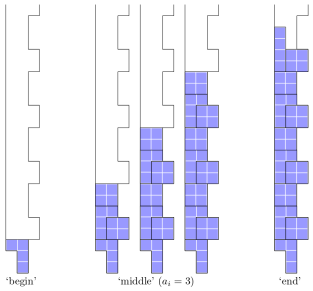
Týmto sme ukázali, že ak sa dajú čísla rozdeliť do trojíc, pozícia Tetrisu sa dá vyhrať. Ostáva ukázať, že ak sa čísla nedajú rozdeliť, pozícia sa nedá vyhrať.

Jamy a postupnosť kociek sú navrhnuté tak, že hráč nemôže jednotlivé kocky (prislúchajúce jednému členu a_i) rozdelovať do rôznych jám – kocky `O`, `J` a `I` jednoducho nemajú ten správny tvar, takže jamu, ktorú nenačneme `L`-kom, by sme zablokovali – úplný dôkaz zahŕňa nudný výpočet všetkých možností: viď obr. 11.3, ktorý dokazuje, že „odpor je zbytočný“ a akýkoľvek pokus odchýliť sa od nami popísaného postupu – t.j. pre každé a_i si zvolím jednu jamu a do nej hádžem všetky kocky – vedie k tomu, že v jame ostanú voľné zablokované políčka (označené ‘×’), ku ktorým sa už nedá dostať a tieto riadky sa už nikdy nezmažú. □

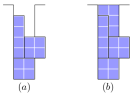
Dá sa ukázať, že Tetris je NP-úplný aj pre hracie plochy fixnej šírky aspoň 8 (Asif a spol., 2019). Hoogeboom a Kosters (2004) dokázali, že ľubovoľná (rozumná) konfigurácia sa dá zostrojiť pomocou vhodnej postupnosti tetromín začínajúc s prázdnou plochou. Zjavne každá kocka má 4 štvorce, takže ak je šírka deliteľná 4, každá dosiahnuteľná konfigurácia bude mať počet štvorcov deliteľný 4. A ak je šírka párna, počet štvorcov bude vždy párny. Samozrejme, žiadna konfigurácia nemôže mať obsadený celý riadok, keďže také riadky miznú. Na druhej strane každá konfigurácia, ktorá spĺňa tieto jednoduché podmienky, sa dá zostrojiť.



Obr. 11.1: Počiatočný stav hracej plochy. Inštancia je zostrojená tak, že kocky tvaru `L`, `O` a `I`, treba porozdelovať tak, aby presne vyplnili s „jám“. Potom príde kocka tvaru `T`, ktorou „odomkneme zámok“ vpravo hore, vrchné dva riadky zmiznú a odkryjú sa voľné stĺpce vpravo. Nasledovať budú už len `I`-čka, ktorými za búrlivých ovácií obecenstva hru vyhráme. Level je zostrojený tak, že sa dá vyhrať práve vtedy, keď sa kocky tetrisu dajú rozdeliť do skupín, aby *presne* vyplnili jednotlivé jamy a to sa dá práve vtedy, keď sa prislúchajúce čísla $a_1, \ldots, a_n$ dajú rozdeliť do s trojíc, pričom každá trojica má súčet T. Takto vieme redukovať problém 3-Partition na Tetris.

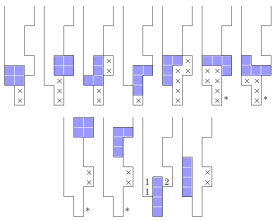


(a) Začneme `L`, potom a_i-krát `O`, `J`, `O`, zakončíme `O`, `I`.

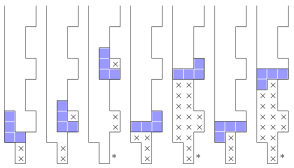


(b) Koncovka – pomocou `L` uzatvoreníme jamu.

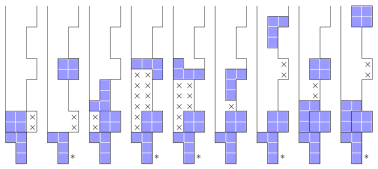
Obr. 11.2: Vypĺňanie jám.



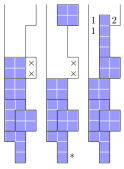
(a) Ak začnem vypĺňať jednu jamu, ale potom skúsim dať `O`, `J`, alebo `I` do inej jamy, nevyhrám.



(b) Ak úvodné `L` uložím inak, ako je „vzorový“ postup, nedopadne to dobre.



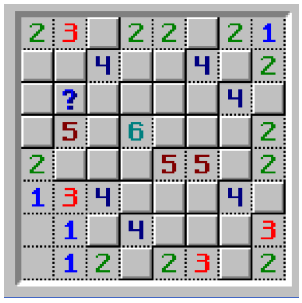
(c) Ak skúsim vypĺňať jamu kockami `O`, `J`, `O` inak, ako bolo zamýšľané, vytvorím si políčka, ktoré sa už nebudú dať vyplniť.



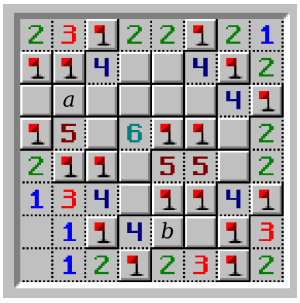
(d) Ak dám záverečné `O`, `I` zle, pokazím si to.

Obr. 11.3: Zlé umiestnenie kociek.

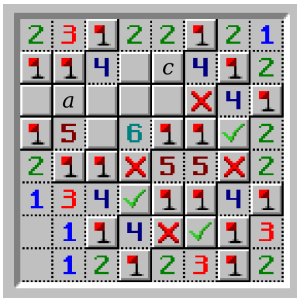
11.2 Hľadanie mín



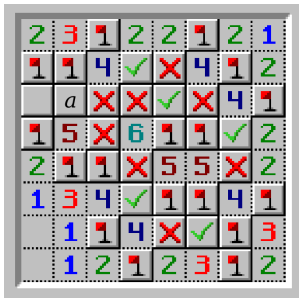
(a) Uvažujme takúto hru. Je na pozícii otáznika mína? Ťažká otázka.



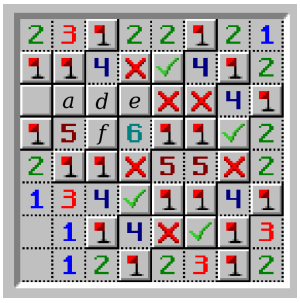
(b) Označme zjavné míny. Pozrime sa na pozíciu b!



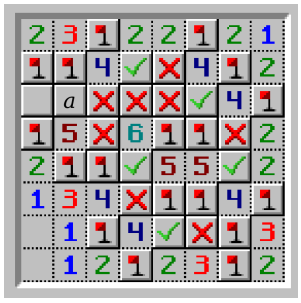
(c) Ak je na b mína, na políčkach × musia byť míny a ✓ sú voľné. Čo c?



(d) Ak je na c mína, potom políčka 4 aj 5 majú potrebný počet mín a a je voľné.



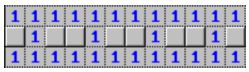
(e) Ak je c voľné, potom 5 a 6 chýbajú dve míny, ale 4-ke chýba iba jedna; míny musia byť na d, f a políčko a je voľné.



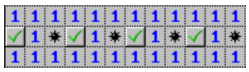
(f) Nakoniec, ak je b voľné, aj a musí byť voľné. Teda a je voľné vo všetkých prípadoch.

Obr. 11.4: Príklad pozície v hre míny.

Označme hodnotu políčka x, pričom $x = 1$ (true) bude znamenať, že na políčku je mína a $x = 0$ (false) bude znamenať, že políčko je voľné.



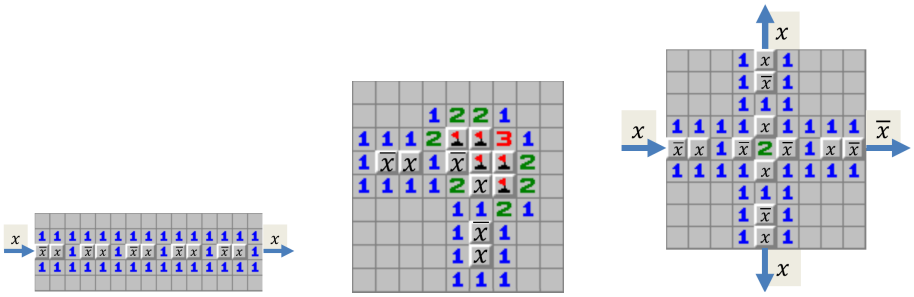
Buď je prvé políčko voľné, na ďalšom je mína, ďalšie je voľné, atď:



alebo je na prvom políčku mína a tento „signál“ sa šíri ďalej:

Ak má prvé políčko hodnotu x, potom ďalšie je $\neg x$, potom x, atď.

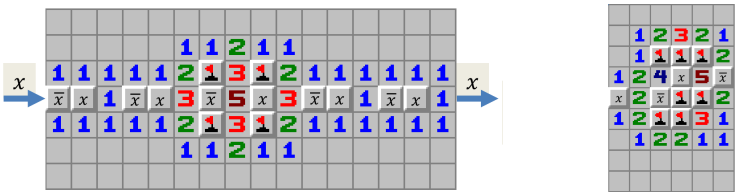
Takýto signál môžeme otáčať aj rozdeliť:



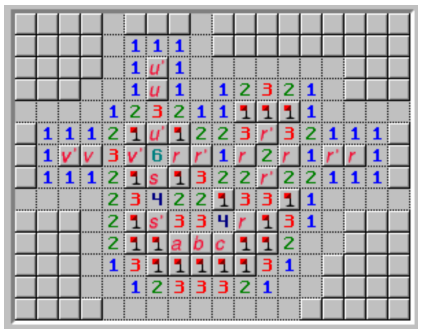
Drôt môžeme začať alebo zakončiť terminálom:

Takže ak zoberieme rozdeľovač a „zaštupľujeme“ horný a dolný koniec, dostaneme hradlo NOT. OR hradlo je na obrázku 11.5 a z OR a NOT dokážeme vytvoriť AND.

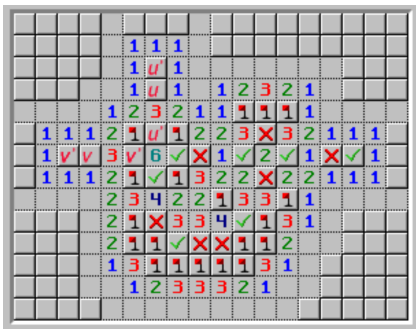
Malá nepríjemnosť, ktorá môže nastať je, že budeme mať problém niektoré gadgety napojiť, lebo budú len trochu mimo (± 1). Všimnite si, že drôt a signál v ňom sa periodicky opakujú vždy po troch políčkach, to znamená, že ak chceme napojiť dva gadgety, musí ich vzdialenosť byť deliteľná tromi.



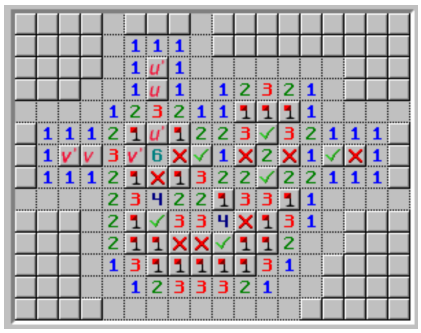
Ak chceme simulovať booleovské obvody, potrebujeme ešte vedieť prekrížiť drôty. To sa dá tiež pomerne jednoducho (skúste navrhnúť takýto gadget), ale v skutočnosti to nie je treba. Ľubovoľný obvod totiž vieme prerobiť na nie o moc väčší planárny. Stačí obvod nakresliť do roviny (pričom sa hrany môžu krížiť) a následne všetky kríženia nahradiť malým obvodom, ktorý vymieňa svoje vstupy, viď obrázok 11.6.



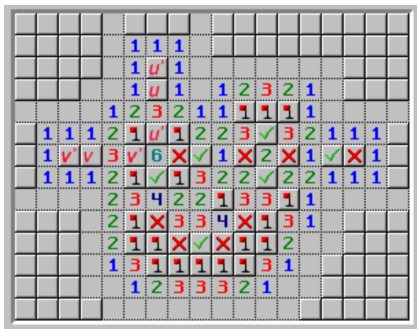
(a) Na políčku r je mína práve vtedy, ak je mína na políčkach u alebo v (alebo oboch).



(b) Postupom odzadu zistíme, že ak políčko r je voľné, tak b aj c musia byť míny, tým pádom je a voľné a u', v' musia byť míny (6-tke chýbajú dve míny). Teda $r = 0$ iba ak $u = v = 0$.

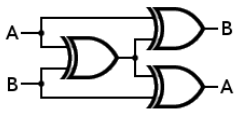


(c) Naopak, ak $r = 1$ je mína, práve jedno z políčok b, c je mína. Sú dve možnosti. Ak je mína na políčku b, s' je voľné. Tento prípad nastane ak sú u' aj v' voľné, teda ak $u = v = 1$, $r = 1$.

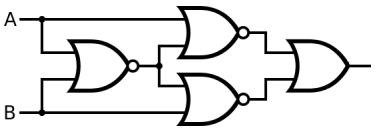


(d) Nakoniec ak $r = 1$, ale mína je na políčku c, tak s je voľné, takže 6-ke chýba práve jedna mína – tento prípad nastane, ak práve jedno z políčok u, v je mína.

Obr. 11.5: OR hradlo.



(a) Výmena pomocou XOR hradiel.



(b) Implementácia XOR pomocou OR a NOT (malé krúžky)

Obr. 11.6: Každé kríženie drôtov vieme nahradiť malým obvodom, kde sa drôty nekrížia a ktorý vstupy vymení. Ide o starý programátorský trik, kde spočítame $A \oplus B$ a následne $A \oplus (A \oplus B) = B$ a $(A \oplus B) \oplus B = A$.

Veta 11.2 (Kaye (2000), Scott a spol. (2011)). *Daná je rozohratá hra míny, úlohou je zistiť, či dané políčko musí byť voľné. Tento problém je* coNP*-úplný. Ekvivalentne problém, či na danom políčku môže byť mína je* NP*-úplný.*

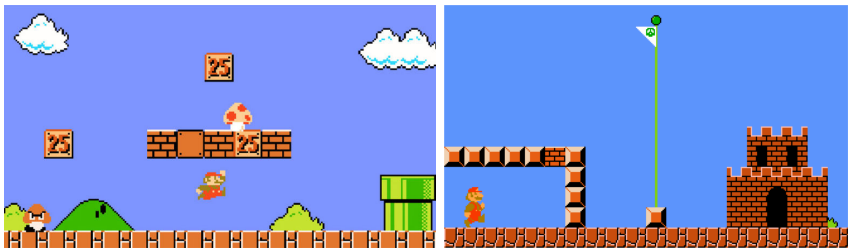
■ **Dôkaz.** Redukciou z nesplniteľnosti planárneho obvodu, čo je coNP-úplný problém zistiť, či pre daný obvod C je $\forall x : C(x) = 0$. Obvod nakreslíme do štvorcovej mriežky tak, aby sa nekrížili hrany; gadgetmi vyššie potom ...

□

11.3 Super Mario

Veta 11.3 (Aloupis a spol. (2015)). *Zistiť, či sa dá level Super Maria vyhrať je* NP*-ťažké.*

■ **Dôkaz.** Redukciou z 3Sat: pre danú formulu vytvoríme úroveň Super Maria, ktorá sa bude dať prejsť práve vtedy, keď je formula splniteľná. Štart a cieľ budú ako na obrázku 11.7. Na začiatku Mario zoberie hríbik, aby bol veľký a veľký musí doraziť aj do cieľa, pretože inak by nedokázal rozbiť tehličky na konci.



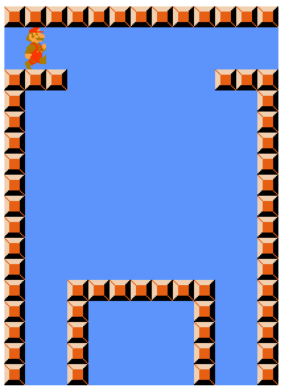
(a) Štart (b) Cieľ

Obr. 11.7

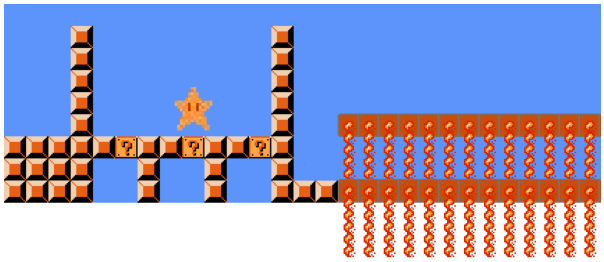
Pre každú premennú budeme mať úsek ako na obrázku 11.8a, kde Mario vojde (či už zľava alebo sprava) a zoskočí dolu tunelom vľavo (true) alebo vpravo (false). Jednotivé plošiny sú dostatočne vysoko, takže na ne Mario nedoskočí a nedokáže sa vrátiť (takto vieme tiež vyrobiť jednosmernú cestu).

Pre každú klauzulu budeme mať úsek ako na obrázku 11.8b. Tri $\boxed{?}$ zodpovedajú trom literálom v klauzule a úroveň je zostrojená tak, že ak je nejaký literál pravdivý, Mario sa vie dostať pod zodpovedajúci $\boxed{?}$ vyboxovať z neho hviezdičku. Na konci budeme potrebovať skontrolovať, že každá klauzula je splnená, čo dosiahneme tak, že na konci bude musieť Mario prejsť cez všetky klauzuly zľava doprava. Napravo od každej klauzuly je plošina s plameňmi a Mario ju dokáže prejsť iba tak, že zoberie hviezdičku, ktorá ho dočasne spraví nesmrteľným. Avšak túto hviezdičku získa iba vtedy, ak sa predtým dostal k aspoň jednému $\boxed{?}$ a teda aspoň jeden literál bol pravdivý.

Celá úroveň bude vyzerať zhruba podľa schémy na obrázku 11.9. Mario musí zo štartu postupne prejsť všetky premenné; v každej premennej x sa rozhodne, ktorou cestou pôjde (či x alebo $\neg x$) a táto cesta potom vedie do všetkých klauzúl, ktoré daný literál obsahujú. V našom príklade cestou x navštívi prvú a druhú klauzulu, alebo cestou $\neg x$ navštívi tretiu klauzulu; v oboch prípadoch potom prejde na ďalšiu premennú, y, kde si opäť musí zvoliť jej pravdivostnú hodnotu. Nakoniec, keď Mario prejde všetky premenné, dostane sa vľavo hore a prejde do cieľa vpravo cez všetky klauzuly. Tie sa dajú prejsť, iba ak v každej vyboxoval hviezdičku nesmrteľnosti, teda iba ak sú všetky klauzuly pravdivé

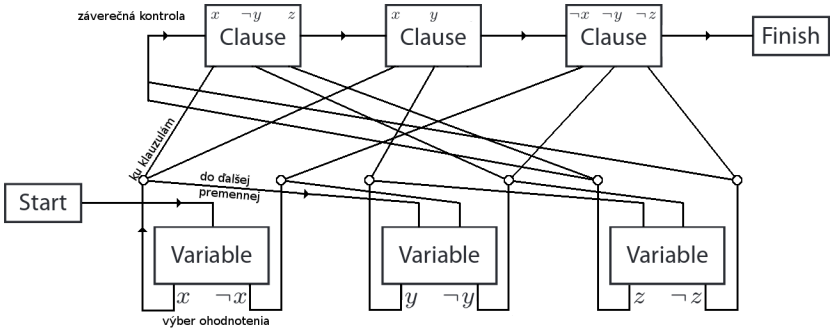


(a) Premenná. Voľba, či zísť vľavo/vpravo zodpovedá nastaveniu premennej true/false.



(b) Klauzula. Mario sa musí vedieť dostať aspoň k jednému ?, z ktorého vybúši hviezdičku. Tá ho dočasne spraví nesmrteľným, aby vedel prejsť ohňom vpravo.

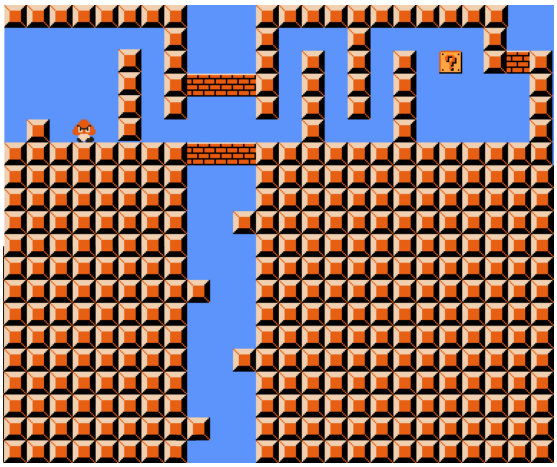
Obr. 11.8



Obr. 11.9: Schéma úrovne v hre Super Mario pre formulu $(x \vee \neg y \vee z) \wedge (x \vee y) \wedge (\neg x \vee \neg y \vee \neg z)$.

Ostáva vyriešiť problém, ako sa budú krížiť chodby. V nákrese schémy na obrázku 11.9 sa totiž niektoré cesty pretínajú, ale pritom predpokladáme, že sa na takejto „križovatke“ nedá jednoducho prejsť z jednej cesty na druhú – v takom prípade by sme sa z jednej premennej mohli dostať aj do iných klauzúl, ktoré daný literál neobsahujú, čo nechceme. Všimnime si však, že nám stačia križovatky, ktoré sú jednosmerné (obojsmerné cesty ku klauzulám vieme nahradiť dvoma jednosmernými tam a späť), stačí, ak sa každá cesta dá prejsť len raz a dokonca môžeme predpokladať, že z každej križovatky buď použijeme len jednu cestu, alebo ak použijeme obe, tak dopredu vieme, v akom poradí. Konkrétne, ak sa krížia dve cesty, ktoré idú z premennej a jej negácie, potom použijeme práve jednu z nich a ak sa krížia dve cesty, ktoré vedú z dvoch rôznych

premenných, tak vieme, ktorú premennú navštívime prvú.



Obr. 11.10: Kríženie. Mario môže prejsť zľava doprava (iba raz) a zdola hore a naspäť. Teoreticky, ak už raz išiel zdola hore a vybúral niektoré tehličky, mohol by pri prechode zľava výsť hore. V našej konštrukcii sa však bude dať ísť buď len jednou cestou, alebo ak obomi, tak najskôr sa pôjde zľava doprava a až potom zdola hore.

Riešnie križovatiek je na obrázku 11.10. Mario ju môže prejsť buď zľava doprava alebo zdola hore. Ak prichádza zľava, nechá sa poraniť zlým hríbikom (volá sa Goomba), čím sa zmenší a tak sa dokáže prepchať cez prekážky až ku ?, kde si vyboxuje nový hríbik, aby bol opäť veľký. Veľký Mario sa cez úzku prekážkovú dráhu nezmestí, takže prechod je možný iba zľava doprava a nie naopak. A ak chce Mario vyjsť vpravo, hríbik musí zobrať, pretože malý Mario nedokáže tehličky vpravo rozraziť. Z rovnakého dôvodu, v strednej časti, keď je malý, si nedokáže preraziť cestu nahor.

Ak Mario prichádza zdola, môže vyhopsať po schodíkoch, rozbiť dolné tehličky vpravo, vyskočiť, rozbiť horné tehličky vľavo a vyskočiť hore. Neskôr sa môže touto cestou vrátiť. Keďže je veľký, neprepchá sa vľavo ani vpravo. Jediné, čo by sa mohlo stať je, že keby Mario najskôr prišiel zdola hore, rozbil tehličky a neskôr by prišiel zľava, mohol by ujsť nahor, ale ako sme spomenuli vyššie, pre každú križovatku vieme, v akom poradí sa cez ňu pôjde a preto ju navrhneme tak, že prvá cesta bude horizontálna a až druhá zvislá. □

Týmto sme dokázali, že problém je NP-ťažký. Čo myslíte, je aj NP-úplný? Patrí tento problém do NP?

Úlohy

- Akú zložitosť má Tetris na konštantne veľkej hracej ploche?

Literatúra

Aloupis, Greg, Erik D Demaine, Alan Guo, a Giovanni Viglietta. 2015. "Classic Nintendo games are (computationally) hard." *Theoretical Computer Science* 586, s. 135–160.

Asif, Sualeh, Erik D Demaine, Jayson Lynch, a Mihir Singhal. 2019. "Tetris is NP-hard even with $O(1)$ columns." In *the 22nd Japan Conference on Discrete and Computational Geometry, Graphs, and Games (JCDCG3 2019)*. s. 37–38.

Breukelaar, Ron, Erik D Demaine, Susan Hohenberger, Hendrik Jan Hoogeboom, Walter A Kosters, a David Liben-Nowell. 2004. "Tetris is hard, even to approximate." *International Journal of Computational Geometry & Applications* 14(01n02), s. 41–68.

Hoogeboom, Hendrik Jan a Walter A Kosters. 2004. "How to Construct Tetris Configurations." *Int. J. Intell. Games & Simulation* 3(2), s. 97–105.

Kaye, Richard. 2000. "Minesweeper is NP-complete." *The Mathematical Intelligencer* 22(2), s. 9–15.

Scott, Allan, Ulrike Stege, a Iris Van Rooij. 2011. "Minesweeper may not be NP-complete but is hard nonetheless." *The Mathematical Intelligencer* 33(4), s. 5–17.

Kapitola 12

PSPACE-ťažké hry

12.1 Prince of Persia, Doom, Quake

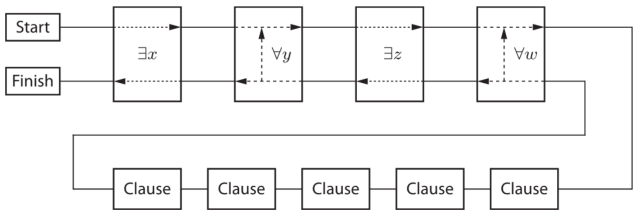
Metaveta 12.1 (Viglietta (2014)). *Ľubovoľná plošinová hra, ktorá obsahuje dvere a „nášľapné plošiny“ (nejaký mechanizmus, ktorý otvára a zatvára dvere), je* PSPACE*-ťažká. Dokonca už len zistiť, či sa dá dôjsť do cieľa je* PSPACE*-ťažké. Predpokladáme pritom, že vieme docieliť, že hráč, ktorý si zvolí cestu s takouto plošinou, sa jej nedokáže vyhnúť (napríklad ju preskočiť alebo obísť) a musí ju aktivovať. Presnejšie: stačí, keď sa nedokáže vyhnúť zatvárajúcim plošinám.*

Dôsledok 12.1. *Hry ako Prince of Persia, Doom, či Quake sú* PSPACE*-ťažké.*

Rozmyslite si, ako umiestniť nášľapné plošiny v týchto hrách tak, aby sa nedali obísť/preskočiť.

■ **Dôkaz.** Redukciou z QBF. Pre danú formulu vytvoríme úroveň, ktorá sa dá prejsť práve vtedy, keď je formula splniteľná. Schéma úrovne je na obrázku 12.1. Hráč postupne prejde cez všetky kvantifikátory; v existenčných musí správne vybrať hodnotu premennej, vo všeobecných kvantifikátoroch je hráč nútený postupne vyskúšať obe hodnoty (najskôr sa nastaví hodnota premennej na true; pri návrate je hráč nútený nastaviť hodnotu na false a znovu prejsť zvyšok formule). Ohodnotenie premenných je reprezentované dverami – ak je $x = 1$, všetky dvere označené x sú otvorené a dvere označené $\bar{x}$ sú zatvorené a naopak, ak $x = 0$, dvere x sú zatvorené a $\bar{x}$ sú otvorené.

Klauzula s tromi literálmi je na obrázku 12.2b. Prejsť sa dá práve vtedy, keď sú aspoň jedny dvere otvorené, t.j. ak je aspoň jeden literál pravdivý a teda ak je klauzula splnená. Kvantifikátory $(\exists x)$ a $(\forall x)$ sú na obrázkoch 12.2c a 12.2d. Keďže jedna plošina otvára len jedny dvere a premenné sa môže nachádzať vo viacerých klauzulách, budeme mať jednu plošinu a dvere pre každý *výskyt* premennej x (značíme $x_1, x_2, \ldots$, respektíve $\bar{x}_1, \bar{x}_2, \ldots$). V existenčnom kvantifikátore si vyberieme jednu cestu (hornú pre $x = 1$ alebo dolnú pre $x = 0$), čím otvoríme príslušné literály a zatvoríme ich negácie. Vyjdeme vpravo, rekurzívne vyhodnotíme zvyšok formule a vrátime sa dolnou cestou sprava doľava.



Obr. 12.1: Schéma úrovne pre QBF formulu tvaru $\exists x \forall y \exists z \forall w : \phi$. Implementácia klauzúl a kvantifikátorov je na obrázku 12.2.

V univerzálnom kvantifikátore najskôr nastavíme hodnotu $x = 1$, ale zavrieme dvere d. Pri návrate musí hráč nastaviť $x = 0$ a znovu obehnúť celý zvyšok formuly. Takto skontrolujeme, že formula je pravdivá pre obe hodnoty x. Až potom môžeme prejsť cez dvere d vľavo.

□

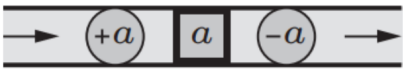
12.2 Super Mario

V predchádzajúcej kapitole sme dokázali, že vyhrať v hre Super Mario je NP-ťažké a pýtali sme sa, či sa to dá rozhodnúť v NP. Odpoveď znie, že asi Nie (ak NP $\neq$ PSPACE) – problém je totiž až PSPACE-úplný!

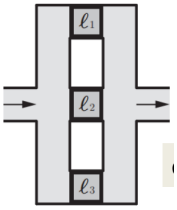
Veta 12.1 (Demaine a spol. (2016)). *Super Mario je* PSPACE*-úplný.*

■ **Dôkaz.** Použijeme schému z predchádzajúceho dôkazu s dverami, ktoré vyzerajú ako na obrázku 12.3a. „Dvere“ v tomto prípade predstavuje ostnatá korytnačka (prezývaná „Spiny“), ktorá stráži a blokuje niektoré cesty. Cesta v ľavej časti označená „traverse“ sa dá prejsť práve vtedy, keď je Spiny v pravej časti (otvorené dvere). Keď však chceme prejsť cez trasu označenú „close“, musíme najskôr korytnačku prehupnúť na druhú stranu. Dosiahneme to tak, že si na ňu počkáme pod tehličkou dolu. Keď sa priblíži, vyskočíme a korytnačku prohodíme na druhú stranu ako na obrázkoch 12.3b–12.3d. Plameň v strede bráni Mariovi prejsť, ale korytnačke neublíži. Takto sa uvoľní cesta „close“, ktorú môžeme prejsť, ale zatvoria sa dvere – cesta „traverse“ je teraz strážená. Otvoriť sa dá tak, že prídeme cestou „open“ a podobne ako pri zatváraní Spinyho prehodíme na pravú časť.

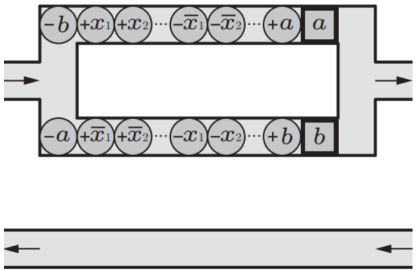
Nášľapné plošiny, ktoré otvárajú/zatvárajú dvere teda „implementujeme“ cestami „open“ a „close“ a dvere sú implementované ako prítomnosť strážiaceho Spinyho. Všimnite si, že pri prechode cestou „open“ Mario môže, ale nemusí otvoriť dvere (to je v poriadku, prejdite si dôkaz metavety ešte raz a presvedšte sa, že neotvoriť dvere mu nijako nepomôže) – dôležité je, že pri prechode cestou „close“ *musí* dvere zatvoriť.



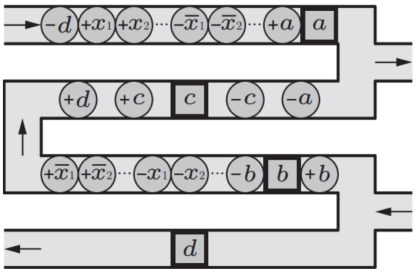
(a) Takáto cesta sa dá prejsť len jedným smerom: ak by sme chceli ísť sprava, nevyhneme sa plošine $-a$, ktorá dvere zatvára.



(b) Klauzula $\ell_1 \vee \ell_2 \vee \ell_3$ – zjavne na to, aby sme prešli, musia byť aspoň jedny dvere otvorené, tzn. klauzula musí byť splnená.



(c) Existenčný kvantifikátor ($\exists x$). Horná cesta zodpovedá x, dolná $\neg x$. Na začiatku cesty najskôr zavrieme opačné dvere, otvoríme všetky dvere, v klauzulách, ktoré zodpovedajú danému literálu a zavrieme všetky negácie. (Keďže jedna plošina otvára/zatvára práve jedny dvere, $x_1, x_2, \ldots$ a $\bar{x}_1, \bar{x}_2, \ldots$ označuje jednotlivé výskyty literálov x, respektíve $\neg x$.)



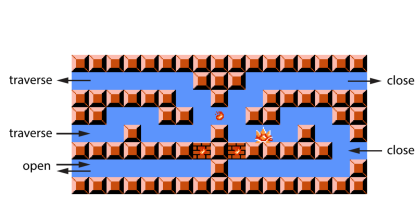
(d) Univerzálny kvantifikátor ($\forall x$). Hráč najskôr zavrie dvere d dolu; nastaví hodnotu x na true a rekurzívne vyhodnotí zvyšok formuly. Pri návrate sa nedostane cez d – musí ísť cez b a c, nastaviť hodnotu x na false (až potom sa otvoria dvere d) a opäť rekurzívne vyhodnotiť zvyšok formuly. Takto zabezpečíme, že princ musí prejsť formulu pre všetky x, true aj false.

Obr. 12.2: Štvorec $\boxed{d}$ označuje dvere, ktoré sa otvárajú plošinou $+d$; naopak $-d$ tieto dvere zatvára. Predpokladáme, že nášľapná plošina sa nedá obísť/preskočiť.

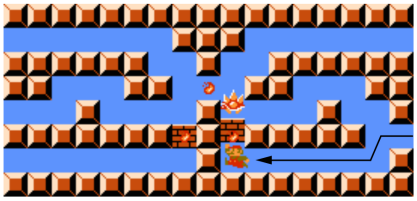
Na rozdiel od plošín ktoré môžeme uložiť kdekoľvek, máme tentokrát jeden gadget s cestami „open“, „close“ a „traverse“ hneď vedľa seba – keď budeme chcieť tieto gadgety pospájať môžu sa nám cesty krížiť a preto na rozdiel od Metavety 12.1 potrebujeme ešte aj križovatky. Najjednoduchšie to je použitím potrubí, ktoré Maria dovedú, kam treba. □

12.3 Geografia

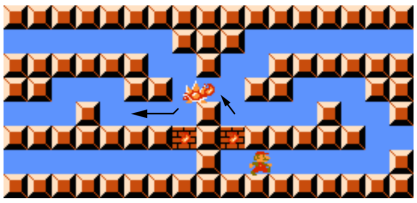
Geografia je jednoduchá slovná hra na cesty, kde hráči striedavo menujú mestá, pričom každé ďalšie mesto musí začínať písmenom, ktorým končí to predošlé



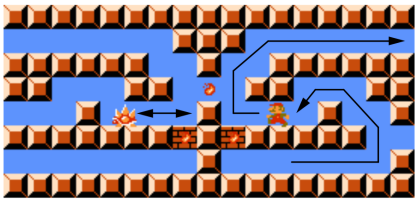
(a) Otvorené dvere: Mario môže prejsť cestou označenou „traverse“.



(b) Ak chce prejsť cestou „close“, musí najskôr prehodiť Spinyho.



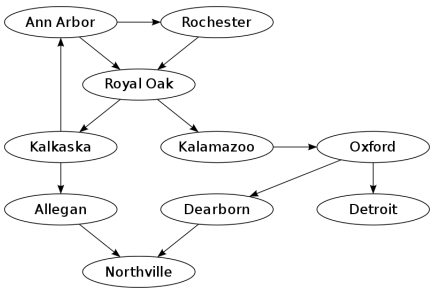
(c) Treba v správnom momente vyskočiť.



(d) Následne môžeme cestou „close“ prejsť, avšak cestu „traverse“ stráži Spiny: dvere sú „zavreté“.

Obr. 12.3: Gadget pre „dvere“ v Super Mariovi.

a mestá sa nesmú opakovať. Túto hru môžeme formulovať ako hru na grafe, kde vrcholy sú názvy všetkých miest a orientované hrany vedú medzi mestami, kde prvé a posledné písmená sedia (pozri obrázok 12.4). Na jednom vrchole je žetón, ktorý hráči striedavo posúvajú po hranách, pričom sa nesmú vrátiť do už navštíveného vrcholu.

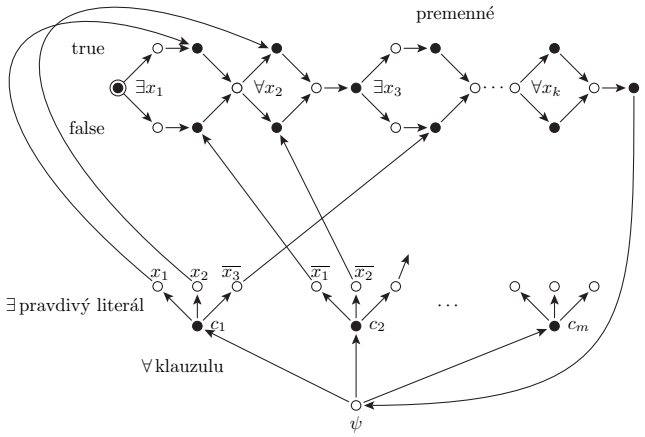


Obr. 12.4: Grafová reprezentácia hry Geografia.

Budeme uvažovať zovšeobecnenú hru, ktorá sa môže hrať na ľubovoľnom grafe. (Rozmyslite si, akú zložitosť má hra, kde graf naozaj zodpovedá názvom miest nad *konečnou* abecedou.) Predpokladáme, že biely položil žetón na označený vrchol, na ťahu je čierny a pýtame sa, či má vyhrávajúcu stratégiu. (Hráčov budeme pre jednoduchosť prezývať „biely“ a „čierny“, podobne ako v šachu, napriek tomu, že v tejto hre figúrky rôznych farieb nie sú.)

Veta 12.2 (Lichtenstein a Sipser (1980)). *Rozhodnúť pre danú pozíciu hry Geografia, či má hráč na ťahu vyhrávajúcu stratégiu, je* PSPACE*-úplný problém.*

■ **Dôkaz.** Redukciou z QBF. Majme formulu v tvare $\phi = \exists x_1 \forall x_2 \exists x_3 \cdots \exists x_k : \psi$. Zodpovedajúci graf bude ako na obrázku 12.5. V existenčných kvantifikátoroch je na ťahu prvý hráč, ktorý vyberá ohodnotenie nepárnych premenných, v univerzálnych kvantifikátoroch je na ťahu druhý hráč a vyberá ohodnotenie párnych premenných. Teda prvý hráč má vyhrávajúcu stratégiu práve vtedy, keď existuje voľba x_1 taká, že pre ľubovoľný ťah súpera x_2 existuje voľba x_3, atď.

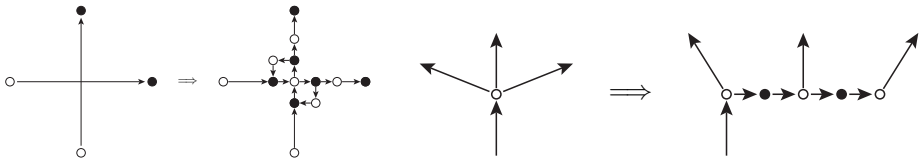


Obr. 12.5: Graf hry Geografia pre danú QBF formulu $\exists x_1 \forall x_2 \exists x_3 \cdots \forall x_k : \psi$. Farba vrcholu zodpovedá informácii, kto je v danej pozícii na ťahu. Začína čierny v zakrúžkovanom vrchole vľavo hore. Hráči postupne nastavia hodnoty premenných $x_1, \ldots, x_k$ – ťah hore = true, dolu = false. Formula je pravdivá práve vtedy, keď je každá klauzla splnená. Čierny vyhrá, ak sa premenné nastavia tak, že nech biely zvolí ľubovoľnú klauzulu, každá je splnená, takže v každej je nejaký literál pravdivý.

Nakoniec, keď hráči zvolia ohodnotenie premenných, treba skontrolovať, či je formula ψ pravdivá, tzn. pre každú klauzulu existuje literál, ktorý je pravdivý. Koncovka vyzerá nasledovne: čierny potiahne do vrcholu ψ, biely vyberie klauzulu (vyhrá, ak nie je splnená) a čierny vyberie literál z klauzuly, ktorý je pravdivý. Z tohto literálu vedie hrana do zodpovedajúceho vrcholu z hornej časti. Ak je literál naozaj pravdivý, v zodpovedajúcom vrchole sme už raz boli – biely teda nemá kam potiahnuť a prehrá. Naopak, ak je literál nepravdivý, vo vrchole sme neboli; biely tam potiahne a čierny prehrá, pretože v nesledujúcom vrchole, kde sa cesty spájajú, sme už raz boli. □

Veta 12.3 (Lichtenstein a Sipser (1980)). *Hra Geografia je* PSPACE*-úplná aj pre planárne bipartitné grafy s vrcholmi stupňa najviac 3.*

■ **Dôkaz.** Upravíme graf, ktorý sme získali v dôkaze predchádzajúcej vety.



(a) Kríženie hrán (vľavo) nahradíme planárnym grafom vpravo.

(b) Vrchol väčšieho stupňa rozbijeme na reťaz, kde každý vrchol má stupeň najviac 3.

Obr. 12.6

Bipartitnosť: Graf na obrázku 12.5 už je bipartitný.

Planárnosť: Každé kríženie hrán nahradíme planárnym podgrafom ako na obrázku 12.6a. Začneme s nakreslením ako na obrázku 12.5, kde sa krížia iba hrany z literálov (posledný ťah bieleho). V každej hre sa použije iba jedna takáto hrana, preto stačí, že v grafe na obr. 12.6a sa dá použiť buď horizontálna alebo vertikálna cesta, ale nie obe.

Maximálny stupeň 3: Každý vrchol väčšieho stupňa rozbijeme na reťaz, kde má každý vrchol stupeň najviac 3 ako na obrázku 12.6b. □

12.4 Reversi

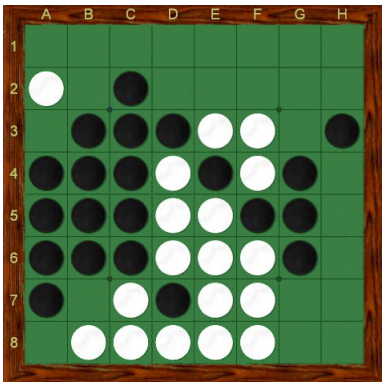
Reversi je dosková hra, ktorá sa hrá na mriežke 8×8 (my budeme uvažovať zovšeobecnenú verziu $n \times n$) s kameňmi, ktoré sú z jednej strany biele a z druhej čierne. Dvaja hráči (biely a čierny) striedavo ukladajú kameň svojej farby na nejaké voľné políčko tak, aby medzi týmto kameňom a iným kameňom tej istej farby v rovnakom riadku, stĺpci, alebo na uhlopriečke bol súvislý neprerušený rad súperových kameňov. Všetky tieto súperove kamene potom obráti, takže sa z nich stanú jeho kamene.

Ak hráč nemôže potiahnuť, pokračuje súper. Ak ani súper nemôže potiahnuť, hra končí. Vyhráva hráč s väčším počtom kameňov.

Veta 12.4 (Iwata a Kasai (1994)). *Problém zistiť, či má v danej pozícii Reversi čierny vyhrávajúcu stratégiu, je* PSPACE*-úplný.*

■ **Dôkaz.** Redukciou z hry Geografia. Ukážeme, ako sa v hre Reversi dá simulovať Geografia tak, že čierny má vyhrávajúcu stratégiu práve vtedy, keď vie vyhrať Geografiu.

Z diaľky bude pozícia v Reversi vyzerať ako na obrázku 12.8. Väčšina hry sa bude odohrávať na malej ploche označenej „simulácia“, kde simulujeme priebeh hry Geografia. Biely má obrovský náskok veľa bielych kameňov v dolnej časti plochy (viď. „teritórium bielych kameňov“). Tieto kamene zaberajú viac ako polovicu celej plochy, takže ak ich ubráni, vyhrá. Naopak, ak mu ich čierny preberie, biely prehrá.



Obr. 12.7: Príklad hry Reversi. Biely na ťahu môže položiť kameň na políčka a3, b2, b7, c1, d2, h4, h5, alebo h6. Súverénne najhoršia voľba je ťah a3 (pri ktorom sa prefarbia kamene b3–d3 a uhlopriečka b4–c5; v nasledujúcom ťahu totiž čierny zahrá a1, čím získa roh, ktorý je strategicky dôležitý – nedá sa prefarbiť. Lepšia možnosť je zahrať napríklad c1 a prefarbiť c-stĺpec.

Ako sa mu to môže podariť? Počas simulácie čierny každým ťahom vytvára hrozby (pozri obrázok 12.9). Prefarbí nejaký „kritický“ kameň, ktorý je spojený radom bielych kameňov až so stĺpcom C_3. Biely ho musí v nasledujúcom ťahu prefarbiť získať naspäť – v opačnom prípade hrozí, že čierny položí kameň do stĺpca C_2, čím prefabí kameň v C_3. Všimnite si, že čierny kameň v stĺpci C_3 sa už nedá prefarbiť. Následne čierny obsadí polia α a α' (prípadne, ak už sú obsadené, tak β, β' alebo γ, γ'), čomu sa nedá zabrániť a následne obsadí pravý dolný roh δ (obr. 12.8. Tým prefarbí časť pravého stĺpca a v nasledujúcich ťahoch postupne riadok po riadku prevezme celé biele územie a vyhrá.

Ak sa čiernemu podarí umiestniť kameň na jedno z políčok α, β, alebo γ, vyhrá. Naopak, ak sa bielemu podarí zabrať políčka α, β, aj γ, vyhrá biely.

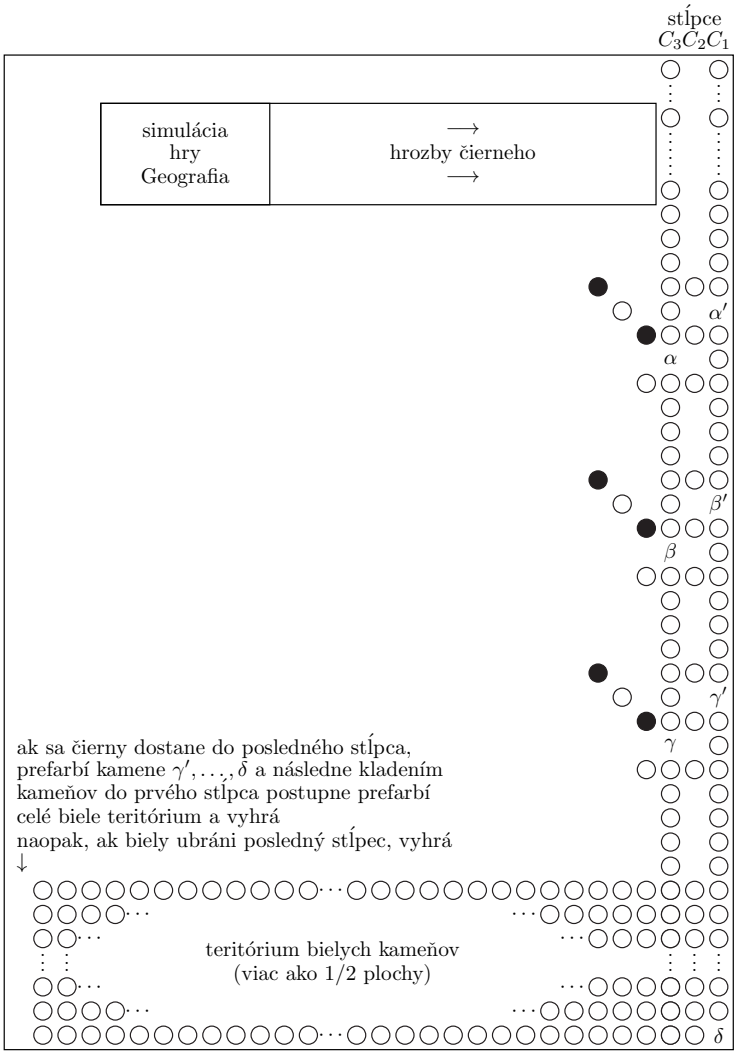
Počas „normálnej“ hry musí čierny každým ťahom vytvárať nové hrozby a biely sa ich musí snažiť odvracať. Ak by sa biely napríklad namiesto „bránenia“ pokúsil „zapchať“ políčka α, β a γ, nestihne to. Tu je príklad možnej výmeny:

1. ... čierny prefarbí kritický vrchol,
2. biely ide na políčko α, čierny do stĺpca C_2,
3. biely na β, čierny na γ a vyhrá

Ak teda čierny vytvorí hrozbu, na ktorú biely nemá odpoveď, čierny vyhrá. Naopak, ak biely všetky hrozby ubráni a čierny nemá ako vytvoriť novú hrozbu, biely stihne zaplniť políčka α, β a γ a vyhrá.

Poďme sa teraz pozrieť, ako simulujeme hru Geografia. Predpokladáme, že graf hry je bipartitný s maximálnym stupňom 3, takže obsahuje len tri typy vrcholov:

1. do vrcholu aj z vrcholu vedie 1 hrana,



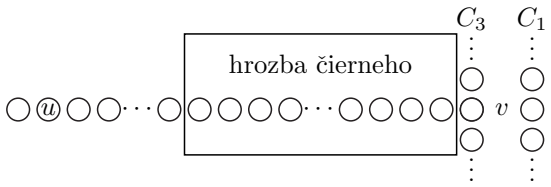
Obr. 12.8: Schéma pozície; väčšina hry sa odohráva na malej ploche označenej „simulácia“. Biely vyhrá, ak ubráni svoje kamene v dolnej časti.

2. do vrcholu vedú 2 hrany, z vrcholu 1,

3. do vrcholu vedie 1 hrana, z vrcholu 2 hrany.

Pre každý typ navyše rozlišujeme, či je na ťahu biely alebo čierny. Gadgety pre rôzne prípady sú na obrázkoch 12.10 a 12.11.

Pri simulácii vstupu do vrcholu v Geografii vždy *aktivujeme* príslušný gadget tak, že na políčko A (alebo A') príde *biely* kameň. Nasleduje postupnosť ťahov na políčka označené $\check{C}_1, B_1, \check{C}_2, B_2, \ldots$, prípadne čiarkované $\check{C}'_1, B'_1, \check{C}'_2, B'_2, \ldots$



Obr. 12.9: Hrozba čierneho: počas simulácie hry Geografia budú biely a čierny prefarbovať tzv. kritické políčka; ak sa napríklad čiernemu podarí prefarbiť políčko u (alebo napravo od u), biely ho musí v nasledujúcom ťahu získať naspäť. V opačnom prípade hrozí, že čierny položí kameň na políčko v, čím prefarbí kameň zo stĺpca C_3. Následne získa jedno z políčok α, β, γ, potom γ', pravý dolný roh δ a vyhrá. Vľavo od u a vpravo od v sú biele kamene, preto biely nevie prefarbiť čierny kameň v C_3 stĺpci.

($\check{C}_i$ sú ťahy čierneho, B_i ťahy bieleho). Výmena spravidla končí tak, že čierny položí kameň na políčko Z (alebo Z' ako záver) a biely zahrá B_3/B_3', čím políčko prefarbí naspäť na biele. Políčko Z je zároveň totožné s políčkom A v nejakom inom gadgete, ktoré sa tým aktivuje – takto hra pokračuje v ďalšom vrchole.

Zdôraznime, že bez ohľadu na to, či je v Geografii na ťahu biely alebo čierny, gadgety v Reversi sa aktivujú vždy bielym kameňom. Keďže graf je bipartitný, vieme, kto má byť v danom vrchole na ťahu a vrcholy z iných partícií reprezentujeme inak. Napríklad pre čierny vrchol s dvoma vstupnými hranami (do ktorého ťahá biely) máme gadget 12.10a, zatiaľčo biely vrchol (do ktorého ťahá čierny) reprezentujeme ako na obrázku 12.10b. V prvom prípade, ak *biely* do vrcholu vstúpi druhýkrát, prehrá. V druhom prípade, ak *čierny* do vrcholu vstúpi druhýkrát, prehrá. Všimnite si, že obr. 12.10b sa od 12.10a líši iba dvoma bielymi kameňmi (označené dvojitým krúžkom) – tie zabezpečia, že v prípade opakovaného návratu vie biely v 12.10b odraziť útok čierneho a vyhrá.

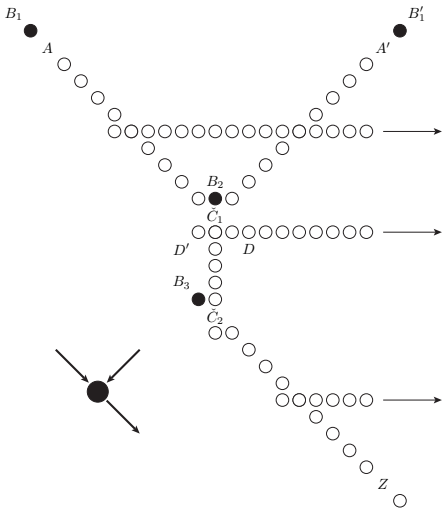
Podobne na obrázkoch 12.10c a 12.10d máme vrchol, z ktorého vychádzajú dve hrany. V bielom vrchole si vyberá biely, v čiernom čierny, ktorým smerom sa bude pokračovať.

Všimnite si, že keďže gadgety „vedia“, kto je na ťahu a aktivujú sa vždy ťahom bieleho, vrcholy s jednou vstupnou a jednou výstupnou hranou nepotrebujeme vôbec reprezentovať. Pri spájaní gadgetov niekedy potrebujeme zmeniť smer (pozri obrázky 12.11a a 12.11b), prípadne paritu pozície (hodnotu $(x + y) \bmod 2$, alebo ak by sme hraciu plochu ofarbili šachovnicovito, niektoré pozície vyjdú na bielych, niektoré na čiernych políčkach) – na to slúži gadget z obrázku 12.11c.

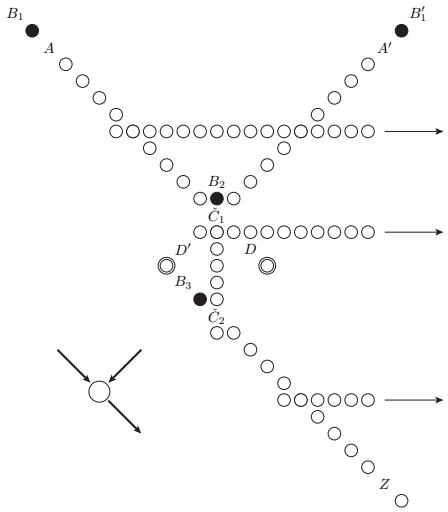
Príklad malej hry Geografia a jej simulácie v Reversi je na obrázku 12.12. □

Ďalšie PSPACE-úplné hry

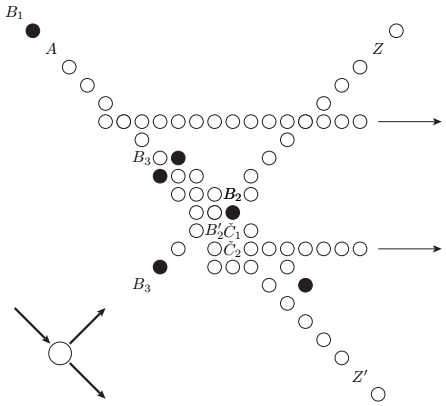
- Sokoban (Culberson, 1997)



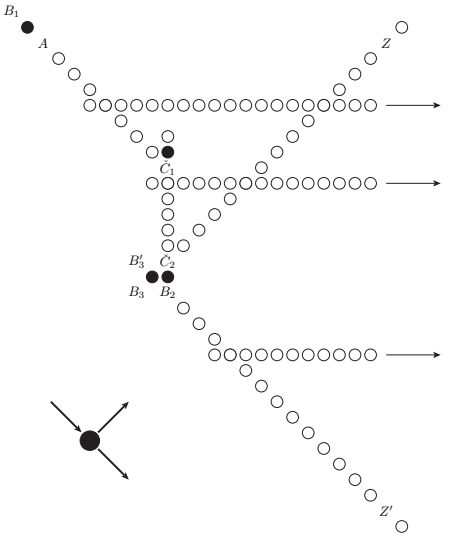
(a) Biely vôjde do vrcholu jednou z dvoch hrán (biely aktivuje políčko A alebo A', na konci je Z biele). Ak vrchol navštívime druhýkrát, čierny vyhrá.



(b) Čierny vôjde do vrcholu jednou z dvoch hrán (biely aktivuje políčko A alebo A', na konci je Z biele); ak ho navštívime druhýkrát, biely vyhrá.

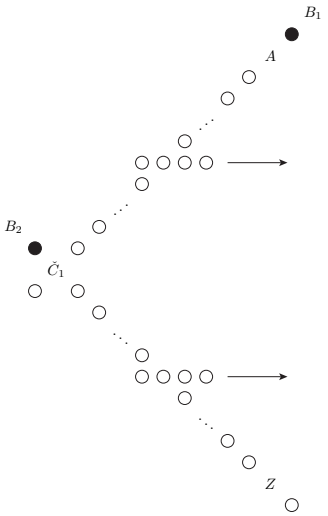


(c) Čierny vôjde do vrcholu, biely si vyberá z dvoch hrán.

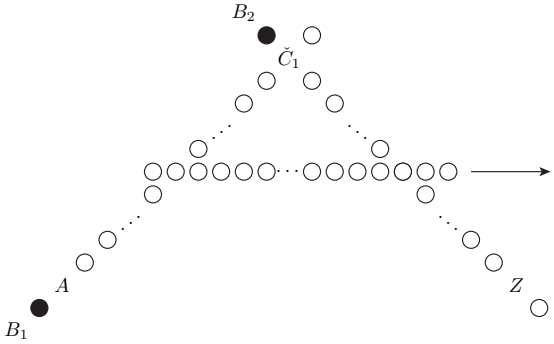


(d) Biely vôjde do vrcholu, čierny si vyberá z dvoch hrán.

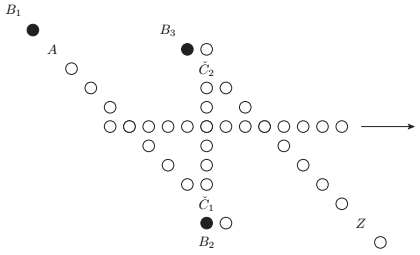
Obr. 12.10: Gadgety na simulovanie Geografie v hre Reversi. Do každého gadgetu vstúpime na políčkach A/A' a vychádzame na políčkach Z/Z'. Medzitým biely ťahá na políčka označené $B_1, B_2, \ldots$ a čierny na políčka označené $\check{C}_1, \check{C}_2, \ldots$ (prípadne s čiarou).



(a) Otočka alebo vrchol s 2 hranami.



(b) Otočka alebo vrchol s 2 hranami.



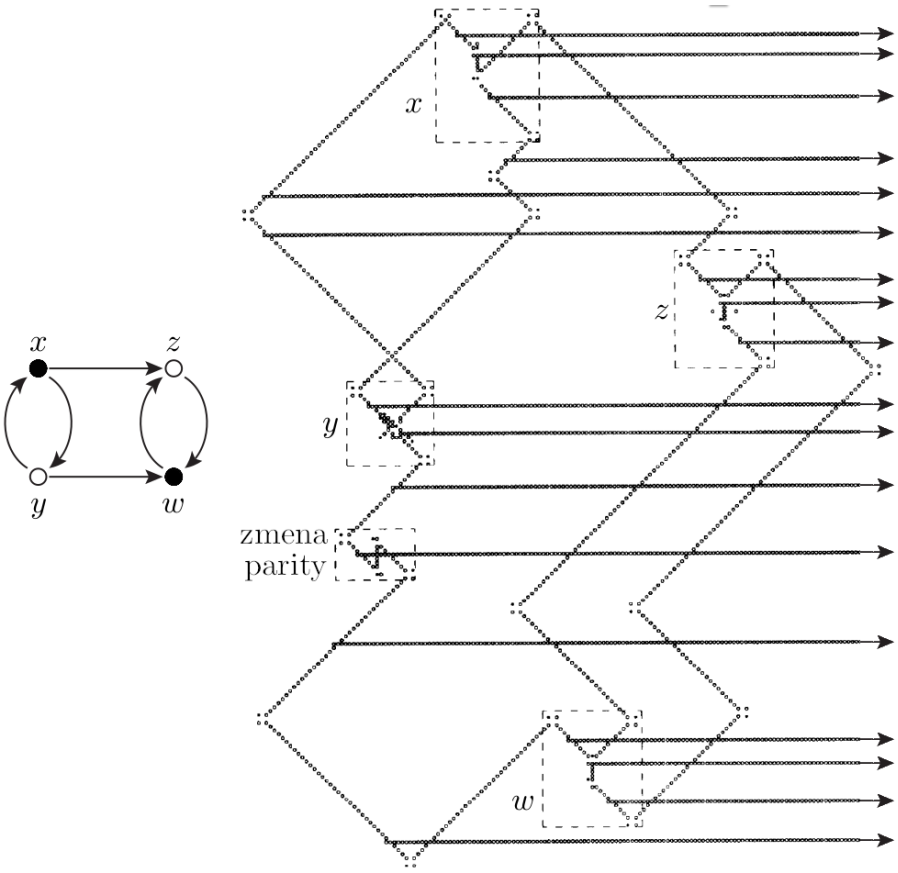
(c) Posun – mení paritu pozície $(x + y)$ mod 2.

Obr. 12.11: Gadgety pre zmenu parity, otočku, alebo vrchol s vstupným aj výstupným stupňom 1.

- Rush Hour (Flake a Baum, 2002)
- Piškvorky (Reisch, 1981)
- Hex a Gomoku (Reisch, 1981)
- Sim (Slany, 2000)
- Amazonky (Hearn, 2005)

Úlohy

- Akú zložitosť má hra Geografia, kde vrcholy sú mestá nad konečnou abecedou?



Obr. 12.12: Príklad grafu pre hru Geografia (vľavo) a zodpovedajúca inštancia hry Reversi. Celá plocha je poskladaná z gadgetov na obrázkoch 12.10 a 12.11.

- Dokážte, že Reversi $\in$ PSPACE.

Literatúra

Culberson, Joseph. 1997. "Sokoban is PSPACE-complete." .

Demaine, Erik D, Giovanni Viglietta, a Aaron Williams. 2016. "Super Mario Bros. is harder/easier than we thought." .

Flake, Gary William a Eric B Baum. 2002. "Rush Hour is PSPACE-complete, or "Why you should generously tip parking lot attendants"." *Theoretical Computer Science* 270(1-2), s. 895–911.

Hearn, Robert A. 2005. "Amazons, Konane, and Cross Purposes are PSPACE-

complete." In *Games of No Chance III, Proc. BIRS Workshop on Combinatorial Games*. s. 287–306.

Iwata, Shigeki a Takumi Kasai. 1994. "The Othello game on an $n \times n$ board is PSPACE-complete." *Theoretical Computer Science* 123(2), s. 329–340.

Lichtenstein, David a Michael Sipser. 1980. "Go is polynomial-space hard." *Journal of the ACM (JACM)* 27(2), s. 393–401.

Reisch, Stefan. 1981. "Hex is PSPACE-complete." *Acta Informatica* 15(2), s. 167–191.

Slany, Wolfgang. 2000. "The complexity of graph Ramsey games." In *International Conference on Computers and Games*. Springer, s. 186–203.

Viglietta, Giovanni. 2014. "Gaming is a hard job, but someone has to do it*"* Theory of Computing Systems *54(4), s. 595–621.*

Kapitola 13

EXP-ťažké hry

V tejto kapitole dáme odpoveď na dve zaujímavé otázky:

- Aké ťažké je hrať dámu?
- Aký ťažký je šach?

V oboch prípadoch nás bude zaujímať problém pre danú pozíciu rozhodnúť, či má hráč vyhrávajúcu stratégiu a vie vynútiť výhru. To je, samozrejme, z pohľadu zložitosti triviálna otázka (hoci sa hovorí, že šach je kráľovská hra), keďže počet všetkých možných pozícií je len konečný. Preto sa budeme zaujímať o zovšeobecnené hry na šachovnici $n \times n$. Ak uvažujeme hry bez pravidla typu „ak sa hrá x ťahov bez výmeny, hra končí remízou“, tak dokážeme, že obe hry, dáma aj šach, sú EXP-úplné.

To je zaujímavé, pretože to znamená, že tieto problémy sa nedajú riešiť v polynomiálnom čase (P $\neq$ EXP). Navyše, ak predpokladáme, že PSPACE $\neq$ EXP, tak existujú pozície, ktoré sa nedajú rozhodnúť v polynomiálnom počte ťahov.

Pri dokazovaní EXP-úplnosti a vôbec uvažovaní o takýchto hrách nám veľmi pomôže charakterizácia cez alternáciu – hráč má vyhrávajúcu stratégiu, ak $\exists$ťah $\forall$ťah súpera $\exists$ťah $\forall$ťah súpera $\cdots \exists$ťah taký, že hráč vyhráva, čo prirodzene zodpovedá alternujúcim TS. Začneme s EXP = APSPACE a prirodzeným EXP-úplným problémom rozhodnúť, či daný alternujúci TS akceptuje slovo v danej pamäti. Odtiaľ prejdeme ako medzikrok ku hrám na boolovských formuliach, kde hrajú dvaja hráči, menia ohodnotenie premenných a snažia sa splniť/nesplniť nejakú formulu. Dokážeme, že ku konkrétnemu ATS vieme zostrojiť formuly tak, že premenné kódujú konfiguráciu stroja a hráči, ktorí nastavujú ohodnotenia premenných simulujú výpočet tohto stroja. Následne si ukážeme, ako sa taká hra na boolovských formuliach dá simulovať v hre dáma, resp. šach.

Skôr než sa však pustíme do dokazovania EXP-ťažkosti, povedzme si pár slov o tom, prečo všetky tieto hry patria do EXP a ako nájsť vyhrávajúcu stratégiu v exponenciálnom čase.

Všetky tieto hry majú exponenciálne veľa možných pozícií a zafunguje úplne všeobecný „ofarbovací“ algoritmus, ktorým dokážeme riešiť ľubovoľnú kombinatorickú hru (dvoch hráčov s perfektnou informáciou bez náhody) v čase úmernom počtu pozícií v hre:

Predpokladajme najskôr, že pozície sa neopakujú a každá hra v konečnom čase skončí výhrou alebo prehrou. Zostrojíme graf všetkých pozícií a začneme ich postupne ofarbovať:

- Ofarbíme všetky koncové pozície – vyhrávajúce aj prehrávajúce.
- Ak z danej pozície vieme potiahnuť do prehrávajúcej (pre súpera!), je vyhrávajúca.
- Ak z danej pozície všetky ťahy vedú do vyhrávajúcich pozícií (pre súpera), je prehrávajúca.

Vyhrávajúca stratégia je zrejmá – ak som vo vyhrávajúcej pozícii, potiahnem tak, aby som súpera dostal do prehrávajúcej; súper nemá inú možnosť, len potiahnuť do pre mňa vyhrávajúcej pozície.

A ako je to v prípade, že môže nastať remíza (napr. pat v šachu, alebo nekonečné opakovanie ťahov)? V skutočnosti na algoritme netreba nič meniť – akurát na konci, keď sa už žiadny vrchol nedá ofarbiť, môžu ostať niektoré vrcholy neofarbené. Všetky tieto pozície sú remízové.

Prečo? Ak má hra N pozícií, môžeme pridať pravidlo, že po N ťahoch sa hra končí remízou – ak si totiž hráč dokáže vynútiť výhru, musí sa to dať bez opakovania pozícií. Môžeme si predstaviť, že algoritmus prebieha vo fázach – najskôr ofarbíme všetky koncové vrcholy, potom tie, z ktorých sa každým ťahom dostaneme do už ofarbeného vrcholu, atď. Potom v i-tej iterácií ofarbíme práve tie pozície, v ktorých hráč na ťahu dokáže vyhrať, alebo nutne prehrá na i ťahov (pri optimálnej hre súpera). Keď sa už nedá žiaden vrchol ofarbiť (maximálne po N-tej iterácií), ofarbili sme všetky pozície, z ktorých jeden hráč dokáže vyhrať na najviac N ťahov (pozri obrázok 13.1).

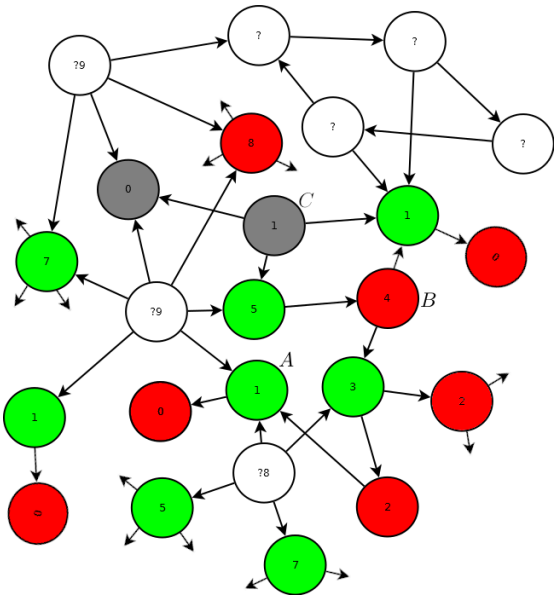
Remízové pozície majú nasledovnú vlastnosť:

- Z remízovej pozície sa dá potiahnuť len do vyhrávajúcich (pre súpera, čo nechceme) a remízových, pričom sa *dá* potiahnuť do remízovej pozície.

Stratégia ako remízovať: ak som v remízovej pozícií, potiahnem do (pre súpera) remízovej pozície (neofarbený vrchol) – ten ma tiež nechce dostať do vyhrávajúcej, takže potiahne opäť do remízovej pozície. Hra sa buď skončí v koncovej remízovej pozícií (pat), alebo sa pozície začnú opakovať a takáto hra predstavuje nekonečnú prechádzku po neofarbených vrcholoch.

13.1 Hry s boolovskými formulami

Stockmeyer a Chandra (1979), ktorí začali štúdium exponenciálne ťažkých hier, vo svojom článku ukázali šesť hier na boolovských formulách, ktoré nazvali



Obr. 13.1: Malá časť grafu všetkých pozícií v hre. Zelenou sme ofarbili pozície, v ktorých hráč na ťahu vyhrá, v červených hráč na ťahu prehrá a šedé sú remíza. Čísla vo vrcholoch označujú minimálny počet ťahov, za ktorý sa dá vynútiť výhra, resp. maximálny počet ťahov, kým hráč prehrá. Napríklad pozícia A je vyhrávajúca, pretože jedným ťahom dostanem súpera do prehrávajúcej. Pozícia B je prehrávajúca – akokoľvek potiahnem, dostanem súpera do vyhrávajúcej pozície; v najlepšom prípade viem prehru oddialiť a prehrám na 4 ťahy; ak by som šiel hore, prehrám ihneď. Pozícia C je remízová – nechcem totiž súpera dostať do vyhrávajúcej pozície vpravo a dolu, ukončím hru patom doľava. Skúste si rozmyslieť, ako zafarbiť zvyšné vrcholy.

jednoducho $G_1, \ldots, G_6$. Na dôkazy zložitosti ďalších problémov nám budú stačiť G_1 a G_3.

Tieto hry hrajú dvaja hráči, ktorých budeme volať „biely“ a „čierny“ (tak ako v šachu). Dané sú dve formuly, biela Φ_1 a čierna Φ_2, nad premennými $X \cup Y$; X je množina bielych, Y množina čiernych premenných. Hráči striedavo menia ohodnotenia svojich premenných (biely mení biele, čierny čierne premenné) a výhra sa určí podľa toho, či je Φ_1, resp. Φ_2 pravdivá.

V hre G_1 sú dané formuly $\Phi_1(X, Y)$ a $\Phi_2(X, Y)$ v 3-CNF. Hráč na ťahu zmení ľubovoľne veľa svojich premenných a ak na konci ťahu nie je jeho formula splnená, prehrá.

Veta 13.1 (Stockmeyer a Chandra (1979)). *Hra G_1 je* EXP*-úplná. Tzn. povedať pre danú pozíciu, či má biely vyhrávajúcu stratégiu, je* EXP*-úplné.*

■ **Dôkaz.** EXP = APSPACE, čiže každý jazyk v EXP má alternujúci TS, ktorý ho akceptuje v polynomiálnej pamäti. Môžeme navyše predpokladať, že tento

stroj je v normálnom tvare, kedy v každom kroku alternuje, – existenčné a univerzálne stavy sa striedajú po každom kroku. Jazyk

$$L = \{\langle M\rangle \# w \mid M \text{ je ATS v normálnom tvare, ktorý akceptuje } w \text{ v pamäti } n\}$$

je EXP-úplný (pamäť stačí lineárna, keďže pri redukcii môžeme vstup polynomiálne nafúknuť).

Tento jazyk redukujeme na G_1. Hra bude zodpovedať výpočtu stroja M, pričom biely vyhrá práve vtedy, keď M akceptuje w. Alternujúci TS M akceptuje, ak existuje taký stav, že pre každý nasledujúci možný stav existuje stav, ..., taký že každý nasledujúci možný stav je akceptačný. To zodpovedá hre: biely vyhrá, ak existuje taký ťah, že pre každý ťah protivníka existuje ťah, ..., taký, že každý nasledujúci ťah je prehrávajúci. Premenné X, resp. Y budú kódovať konfigurácie C_X, resp. C_Y. Hráči budú striedavo voliť ďalšiu konfiguráciu, pričom formuly budú kontrolovať, že M sa naozaj vie dostať z jednej konfigurácie do ďalšej. $\Phi_1(X,Y)$ kontroluje, či $C_Y \vdash_M C_X$, $\Phi_2(X,Y)$ zase kontroluje, či $C_X \vdash_M C_Y$. Tieto formuly vieme zostrojiť ako v Cook-Levinovej vete. □

V hre G_3 sú formuly $\Psi_1(X,Y)$ a $\Psi_2(X,Y)$ v 8-DNF. Hráč na ťahu musí zmeniť *práve jednu* premennú a ak je na konci ťahu splnená jeho formula, *prehrá*. To znamená, hráči sa snažia *nesplniť* svoju formulu (súvisí to s prechodom z CNF na DNF).

Veta 13.2 (Stockmeyer a Chandra (1979)). *Hra G_3 je* EXP*-úplná. Rovnako variant, kde sa hráč môže vzdať ťahu a nepotiahnuť je* EXP*-úplný.*

■ **Dôkaz.** Redukciou z G_1; jeden ťah v G_1, ktorý zmení ľubovoľne veľa premenných „odsimulujeme“ v hre G_3 viacerými ťahmi. Konkrétne, nech G_1 má m bielych a m čiernych premenných. V G_3 im bude zodpovedať $2\times(2m+2)$ premenných, označme ich $x_1,\ldots,x_{2m+2}, x'_1,\ldots,x'_{2m+2}$ a $y_1,\ldots,y_{2m+2}, y'_1,\ldots,y'_{2m+2}$.

$$\underbrace{\begin{matrix} x'_1,\ldots,x'_m, \\ x_1,\ldots,x_m, \end{matrix}}_{\substack{\text{zodpovedá} \\ x_1,\ldots,x_m \text{ v } G_1}} \quad \underbrace{\begin{matrix} x'_{m+1}, \\ x_{m+1}, \end{matrix}}_{\substack{\text{koniec} \\ \text{B ťahu}}} \quad \underbrace{\begin{matrix} x'_{m+2},\ldots,x'_{2m+1}, & x'_{2m+2} \\ x_{m+2},\ldots,x_{2m+1}, & x_{2m+2} \end{matrix}}_{\substack{\text{pomocné} \\ \text{premenné}}}$$

$$\underbrace{\begin{matrix} y'_1,\ldots,y'_m, \\ y_1,\ldots,y_m, \end{matrix}}_{\substack{\text{pomocné} \\ \text{premenné}}} \quad \underbrace{\begin{matrix} y'_{m+1}, \\ y_{m+1}, \end{matrix}}_{\substack{\text{pomocné} \\ \text{premenné}}} \quad \underbrace{\begin{matrix} y'_{m+2},\ldots,y'_{2m+1}, \\ y_{m+2},\ldots,y_{2m+1}, \end{matrix}}_{\substack{\text{zodpovedá} \\ y_1,\ldots,y_m \text{ v } G_1}} \quad \underbrace{\begin{matrix} y'_{2m+2} \\ y_{2m+2} \end{matrix}}_{\substack{\text{koniec} \\ \text{Č ťahu}}}$$

Premenné $x_1,\ldots,x_m$ a $y_{m+2},\ldots,y_{2m+1}$ budú zodpovedať premenným v G_1, zvyšné sú pomocné. Zariadime to tak, že premenné sa budú meniť dokola postupne zľava doprava, pričom v i-tom ťahu biely zmení x_i alebo x'_i a čierny zmení y_i alebo y'_i. Každých $2m+2$ ťahov zodpovedá jedinému ťahu bieleho a čierneho v G_1. Ak sa hráč rozhodne zmeniť x_i (y_{m+1+i}), zodpovedá to zmene i-tej premennej v G_1; ak sa namiesto toho rozhodne zmeniť čiarkovanú premennú

x'_i (y'_{m+1+i}), zodpovedá to ponechaniu i-tej premennej v G_1. Ťahy $1, \ldots, m$ zodpovedajú ťahu bieleho – biely nastavuje biele premenné, zatiaľčo čierny mení pomocné premenné, ktoré nemajú žiaden význam. Následne zmena x_{m+1}/x'_{m+1} zodpovedá koncu ťahu bieleho. Počas ťahov $m+2, \ldots, 2m+1$ čierny nastavuje svoje premenné (simulujeme ťah čierneho), zatiaľčo biely mení svoje pomocné premenné. Nakoniec zmena y_{2m+2}/y'_{2m+2} zodpovedá koncu čierneho ťahu.

Formula Ψ_1 sa bude skladať jednak z negácie Φ_1 (negácia 3-CNF bude 3-DNF formula; splnenie negácie zodpovedá nesplneniu Φ_1) a jednak z častí, ktoré zabezpečia, že hra G_3 sa bude naozaj hrať tak, ako sme si práve popísali a ak v i-tom kroku hráč nezmení premennú z dvojice x_i alebo x'_i, prehrá (analogicky pre Ψ_2 a čierneho hráča).

Všimnime si hodnoty $a_i = x_i \oplus x'_i$. Počiatočnú pozíciu nastavíme tak, že sú to samé nuly. Postupne, keď sa v i-tom kroku zmení práve jedna z x_i a x'_i, hodnota a_i sa preklopí (zvyšné ostanú nezmenené). Vektor $\vec{a}$ teda začína na $\vec{0}$, potom sa zľava doprava začnú pridávať jednotky a po i-tom ťahu by $\vec{a}$ mal byť v tvare 1^i0^*, až na konci kola by a_i mali byť samé jednotky. V nasledujúcom kole sa zase naopak zľava doprava začnú preklápať jednotky na nuly a $\vec{a}$ by mal byť v tvare 0^*1^*.

Ak teda zoxorujeme dve po sebe idúce hodnoty, $A_i = a_i \oplus a_{i-1}$ pre $i > 1$, ale $A_1 = (a_1 \equiv a_{2m+2})$, tak hodnoty A_i detegujú prechod medzi blokom núl a blokom jednotiek. Podobne definujeme $b_i = y_i \oplus y'_i$ a $B_i = b_i \oplus b_{i-1}$, $B_1 = (b_1 \equiv b_{2m+2})$; pozri obrázok 13.3. Pred i-tym ťahom bieleho by $\vec{a}$ aj $\vec{b}$ mali byť v tvare $1^{i-1}0^*$ alebo $0^{i-1}1^*$, takže $A_i = B_i = 1$ a všetky ostatné hodnoty by mali byť nula. Podobne pred i-tym ťahom čierneho by malo byť $A_{i+1} = B_i = 1$ (keďže biely už potiahol) a všetky ostatné A_j, B_j by mali byť nulové. Presne toto budú kontrolovať formuly Ψ_i:

$$\text{nelegit}_1 = \bigvee_{j \neq k} (A_j \wedge A_k) \vee \bigvee_j (B_j \wedge \neg A_{j+1})$$

$$\text{nelegit}_2 = \bigvee_{j \neq k} (B_j \wedge B_k) \vee \bigvee_j (A_j \wedge \neg B_j)$$

Všimnime si, že na začiatku je $A_1 = B_1 = 1$ a ostatné hodnoty nulové. Prvá časť formúl zabezpečí, že hodnoty A_j (resp. B_j) obsahujú vždy najviac jednu jednotku (a teda práve jednu jednotku) a teda hodnoty a_j (resp. b_j) sa skladajú najviac z dvoch súvislých blokov núl a jednotiek. Druhá časť zabezpečí, že ak pred ťahom bieleho bolo $A_i = B_i = 1$, rozhranie blokov je medzi $i-1$ a i a biely musí zmeniť x_i alebo x'_i, aby bolo $A_i = 0$, $A_{i+1} = 1$.

Nakoniec formuly Ψ_1, Ψ_2 definujeme nasledovne:

$$\Psi_1 = \text{nelegit}_1 \vee (A_{m+1} \wedge \neg\Phi_1(x_1, \ldots, x_m, y_{m+2}, \ldots, y_{2m+1}))$$

$$\Psi_2 = \text{nelegit}_2 \vee (B_{2m+2} \wedge \neg\Phi_2(x_1, \ldots, x_m, y_{m+2}, \ldots, y_{2m+1}))$$

Hráč prehrá, ak spraví nelegitímny ťah, alebo ak je na konci svojho ťahu (A_{m+1}, resp. B_{2m+2}) a nie je splnená formula Φ_i z G_1. Samozrejme, hodnoty

a_i	$\cdots$	0	0	1	1	1	$\cdots$
b_i	$\cdots$	0	0	1	1	1	$\cdots$
A_i	$\cdots$	0	0	1	0	0	$\cdots$
B_i	$\cdots$	0	0	1	0	0	$\cdots$

(a) Typický stav premenných pred ťahom bieleho.

a_i	$\cdots$	0	0	0	1	1	$\cdots$
b_i	$\cdots$	0	0	1	1	1	$\cdots$
A_i	$\cdots$	0	0	0	1	0	$\cdots$
B_i	$\cdots$	0	0	1	0	0	$\cdots$

(b) Typický stav premenných pred ťahom čierneho.

Obr. 13.3: Typický stav premenných – $\vec{a}$ a $\vec{b}$ sú v tvare 0^*1^* alebo 1^*0^*; tým pádom $\vec{A}$ a $\vec{B}$ obsahujú práve jednu jednotku – na rozhraní bloku núl a jednotiek, tam, kde treba potiahnuť.

A_i, B_i, ktoré sme definovali cez $\oplus$ treba prepísať pomocou $\wedge, \vee, \neg$ a výsledné formuly roznásobiť a upraviť do 8-DNF. □

13.2 Blok

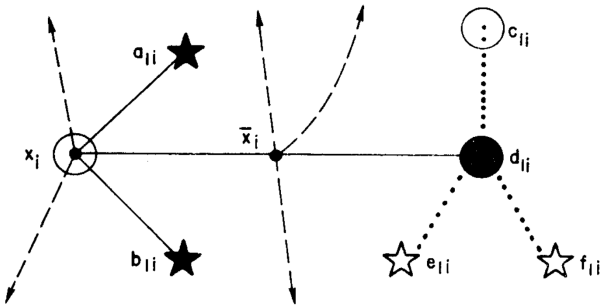
Skôr ako prejdeme k dáme a šachu, ukažme si ešte jednu hru so žetónmi na grafe.

Blok je hra pre dvoch hráčov (volajme ich biely a čierny). Hrá sa na grafe, kde hrany sú ofarbená jednou z troch farieb (na obr. sú to plné, prerušované a bodkované čiary) a niektoré vrcholy sú označené ako cieľové (na obr. biele a čierne hviezdičky). Každý hráč má žetóny svojej farby a snaží sa „skórovať“ – dostať aspoň jeden svoj žetón do cieľa svojej farby. Prvý, komu sa to podarí, vyhráva. Samozrejme, protihráč sa okrem skórovania snaží blokovať žetóny súpera – odtiaľ názov hry. Hráči sa striedajú a hráč na ťahu si vždy vyberie jeden svoj žetón a posunie ho na iný vrchol, pričom v jednom ťahu môže ísť iba pozdĺž cesty jednej farby a nemôže sa zastaviť na ani prejsť cez vrchol, na ktorom už nejaký žetón je.

Veta 13.3 (Stockmeyer a Chandra (1979)). *Hra Blok je* EXP*-úplná.*

■ **Dôkaz.** Redukciou z G_3. Gadget pre (bielu) premennú je na obr. 13.4. Gadget pre čiernu premennú vyzerá podobne, iba farby cieľov a žetónov sú vymenené. Pozície x_i a $\overline{x_i}$ zodpovedajú ohodnoteniu premennej x_i. Väčšinu hry bude biely presúvať svoje žetóny medzi x_i a $\overline{x_i}$ (pre nejaké i), čím simuluje ťah v hre G_3. Všimnite si, že biely svojím žetónom na x_i alebo $\overline{x_i}$ nemôže potiahnuť inak (napr. pozdĺž prerušovanej hrany), v opačnom prípade čierny žetón z d_{1i} skóruje buď na a_{1i} alebo b_{1i}. Naopak, čierny nemôže pohnúť žetónom z d_{1i} (a napr. zablokovať bielemu vrchol $\overline{x_i}$), pretože potom biely skóruje jedným ťahom z c_{1i} na e_{1i} alebo f_{1i}.

Pre každú klauzulu máme gadget ako na obr. 13.5. Celý graf sa teda skladá z jedného gadgetu pre každú premennú a jedného gadgetu pre každú klauzulu, pričom vrcholy označené ako $x_i/\overline{x_i}$, resp. $y_j/\overline{y_j}$ (pre čiernu premennú) na oboch obrázkoch označujú ten istý vrchol. Prerušované hrany na obr. 13.4 z x_i a $\overline{x_i}$ vedú do gadgetov pre klauzuly, kde sa tieto premenné nachádzajú; na obr. 13.5



Obr. 13.4: Gadget pre bielu premennú. Pozície x_i a $\overline{x_i}$ zodpovedajú ohodnoteniu premennej x_i a väčšinu hry bude biely počas svojho ťahu len presúvať takéto žetóny medzi x_i a $\overline{x_i}$ (pre nejaké i).

je zase na vrcholy x_3, resp. $\overline{y_5}$ napojený gadget pre tieto premenné (a ďalšie klauzuly, kde sa tieto premenné vyskytujú).

Na obr. 13.5 je príklad pre čiernu klauzulu $\overline{x_3} \wedge y_5$. Ak je po ťahu čierneho nejaká jeho klauzula splnená, v hre G_3 to znamená, že prehral. Ukážme si, ako v tomto prípade biely dovedie žetón do cieľa. Ak je klauzula na obr. 13.5 splnená, žetóny sú na pozíciach $\overline{x_3}$ a y_5 a to znamená, že vrcholy x_3 a $\overline{y_5}$ na obrázku *sú voľné*. Akonáhle čierny spôsobí, že klauzula je pravdivá, biely vyštartuje žetónom z vrcholu w do s_1, čím spustí záverečnú postupnosť ťahov. Čierny musí odpovedať $v \to r_1$, aby zablokoval cestu k cieľu vľavo. Ak je klauzula naozaj splnená, vrcholy x_3 a $\overline{y_5}$ nie sú blokované a biely môže potiahnuť $s_1 \to x_3 \to u_1$. Čierny musí blokovať cieľ vľavo: $r_1 \to t_1$. Biely ide do s_2, čierny blokuje r_2, biely ide do u_2 a ak čierny blokuje t_2, skóruje vo vrchole z.

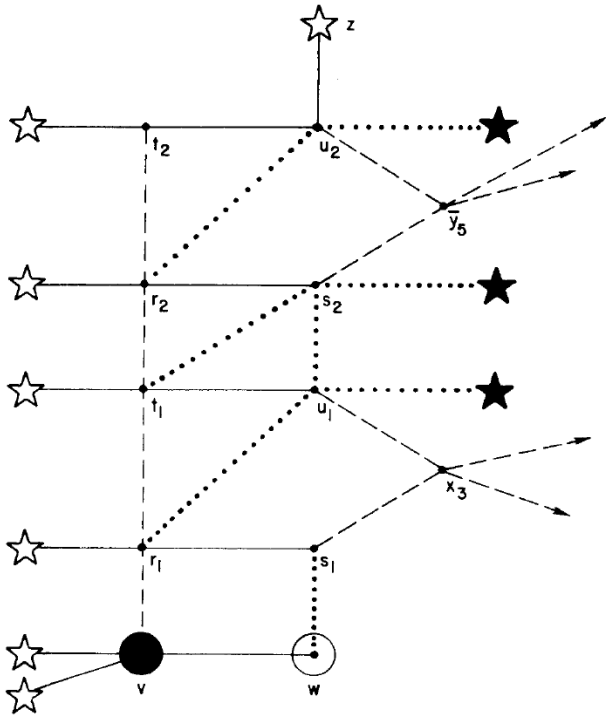
Táto fáza hry zodpovedá overeniu, že daná klauzula je naozaj splnená. Všimnite si tiež, že ak biely začne overovať klauzulu, ktorá nie je splnená, prehrá. Po ťahoch $w \to s_1$ a $v \to r_1$ sa biely *musí* vedieť dostať do u_1 – v opačnom prípade čierny skóruje pozdĺž bodkovanej cesty. Po $s_1 \to u_1$ a $r_1 \to t_1$ sa biely musí vedieť dostať do s_2, v opačnom prípade čierny opäť skóruje po bodkovanej ceste.

Väčšinu hry budú hráči strategicky posúvať žetóny po premenných a meniť ich ohodnotenie. Keď raz niektorý hráč začne overovať nejakú klauzulu, niet návratu a hra po pár ťahoch končí – buď bola klauzula pravdivá a vyhrá, alebo nebola a prehrá. □

13.3 Dáma

Po svete existuje veľa variácií tejto hry, takže si na začiatku pripomeňme a dohodnime pravidlá. Tu sa budeme zaoberať konkrétne verziou známou ako Anglická dáma.

V dáme sa figúrky pohybujú diagonálne na voľné políčka alebo skáču cez



Obr. 13.5: Gadget pre čiernu klauzulu $\overline{x_3} \wedge y_5$. Ak je klauzula splnená, čierny prehrá: žetóny sú vtedy vo vrcholoch $\overline{x_3}$ a y_5, takže vrcholy x_3 a $\overline{y_5}$ nie sú blokované a dá sa cez ne prejsť; biely žetón vyštartuje z w a skóruje buď do niektorého cieľa vľavo, alebo do z hore – čierny nemá šancu ho vyblokovať.

cudzie figúrky a tak ich vyhadzujú. Ak sa pešiak dostane na posledný rad, premení sa na dámu.

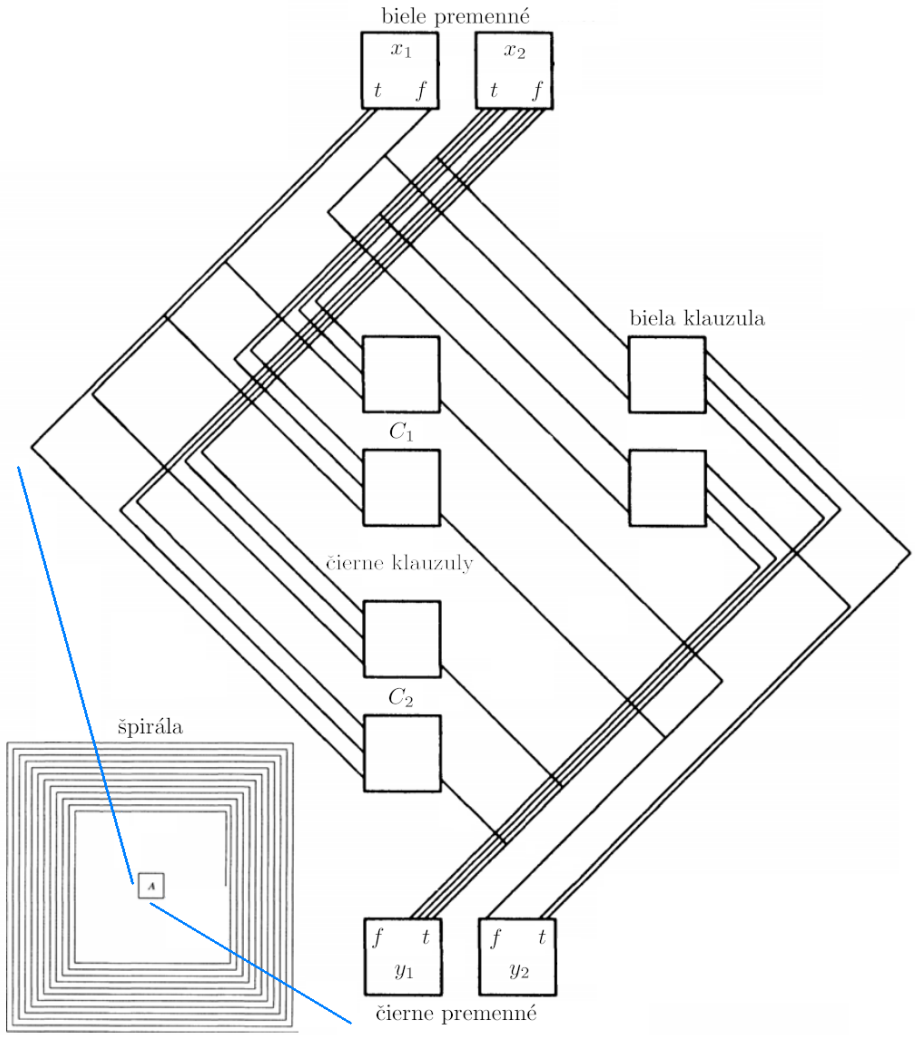
- Všetky figúrky (pešiaci aj dámy) sa pohybujú *po jednom políčku* diagonálne, pričom pešiaci môžu ísť *len dopredu*, dámy aj dozadu. (V iných verziách hry môže dáma prejsť ľubovoľne veľa políčok – v našej nie.)

- Ak figúrka susedí diagonálne s figúrkou súpera a políčko za ňou je voľné, môže ju preskočiť a tak ju zobrať. Ak aj nová pozícia susedí s figúrkou súpera, v skákaní musí pokračovať. Dámy môžu skákať dopradu aj dozadu, ale vždy len o dve políčka, pešiaci môžu skákať iba dopredu.

- Ak hráč na ťahu má možnosť skákať, *musí skákať* a pokračovať v skákaní, kým sa dá. (Toto je rozdiel napr. oproti slovenskej verzií, kde figúrku, ktorá neskákala môže súper zobrať.)

Veta 13.4 (Robson (1984)). *Hra Anglická dáma je* EXP*-úplná.*

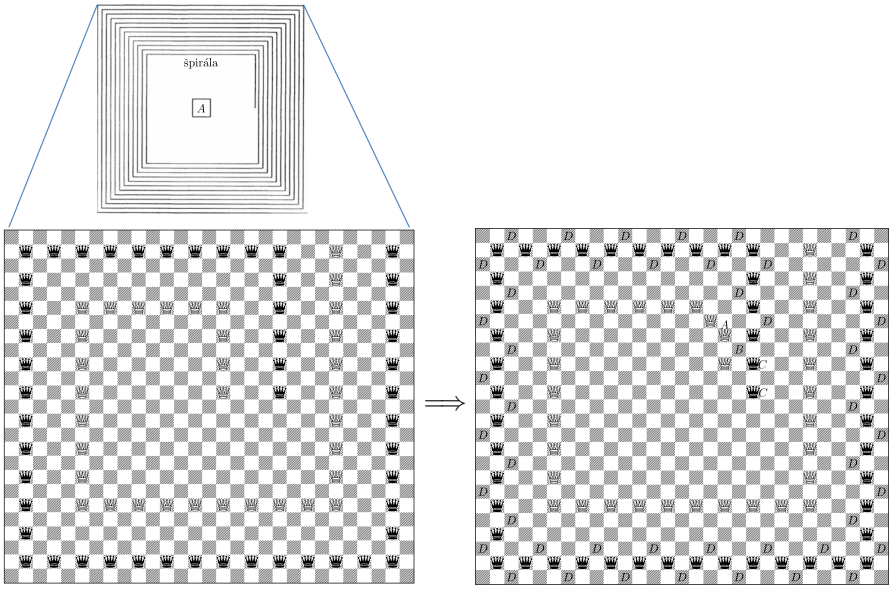
■ **Dôkaz.** Redukciou z G_3. Ukážeme si, ako k daným formulám vytvoriť pozíciu v hre dáma. Schématicky je znázornená na obr. 13.6. Väčšina hry sa bude odohrávať na malej ploche obkolesenej špirálou so strašne veľa bielymi a čiernymi dámami.



Obr. 13.6: Schéma pozície simulujúcej hru G_3. V gadgetoch pre premenné sa dá prepínať medzi hodnotami true/false; premenné sú spojené s klauzulami – ak je nejaká splnená, súper ju „aktivuje“ a dovedie hráča k „spomaľovačom“, kde bude strácať ťahy. Celá táto konfigurácia je uprostred veľkej špirály dám, o ktorú sa hrá (viď obrázok 13.7).

Hlavná myšlienka využíva pravidlo, že hráč, ktorý má možnosť skákať, musí

skákať – a to aj v prípade, že je to preňho nevýhodné. V konštruovanej pozícií sa hráč snaží spustiť takzvané spomaľovače, kde súper síce zoberie zopár bezvýznamných figúrok, ale musí ich vziať, hoci je to plytvanie ťahmi a za ten čas sa hráčovi podarí vyžrať celú špirálu. Tento postup je znázornený na obr. 13.7: Kým čierny berie bezvýznamných pešiakov, biely presunie dámy z pozície na obr. 13.7a do pozície na obr. 13.7b, následne pohne dámu z pozície A na B, čierna dáma na pozícii C ju musí preskočiť, čím sa dostane na políčko A a dáma nad ňou ju preskočí a pokračuje v skákaní po políčkach označených D, pričom požerie skoro všetky čierne dámy v špirále. Následne biely preskupí svoje dámy do súvislých obdĺžnikov a dá sa dokázať, že ak je týchto obdĺžnikov viac ako všetkých čiernych figúrok vnútri, biely dokáže zaručene vyhrať. (A podobne naopak, ak by biely musel skákať cez figúrky v spomaľovačoch, dokáže čierny medzičasom popresúvať svoje dámy a vyžrať celú bielu časť špirály.)



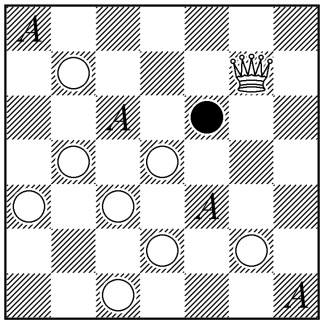
(a) Kto z koho? Väčšina hry sa odohráva na malom území (A) obkolesenom špirálou. Kto vyhrá boj o túto špirálu, vyhrá celú hru.

(b) Ak čierny stratí čas braním bezvýznamných figúrok v spomaľovačoch, biely preskupí svoje dámy a vyžerie celú špirálu.

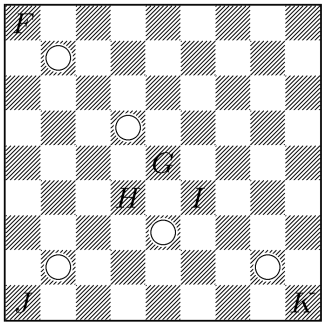
Obr. 13.7: Špirála: Ten, kto vyhrá boj o špirálu, vyhrá celú hru.

Biely spomaľovač je znázornený na obr. 13.8a – ak čierny preskáče pozície označené A a zoberie pešiakov medzi nimi, bielej dáme na pozícií B sa uvoľní políčko a musí skákať. Použitím viacerých kópií vieme bieleho dostatočne spomaliť. Rozmyslite si, ako by mohol vyzerať čierny spomaľovač.

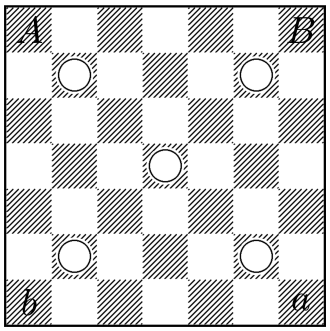
Pre každú bielu premennú máme gadget ako na obr. 13.9. Pre čierne premenné treba obrázok preklopiť a vymeniť farby. Väčšinu hry budú biely aj čierny iba presúvať dámy medzi políčkami T a F (true/false), čo zodpovedá



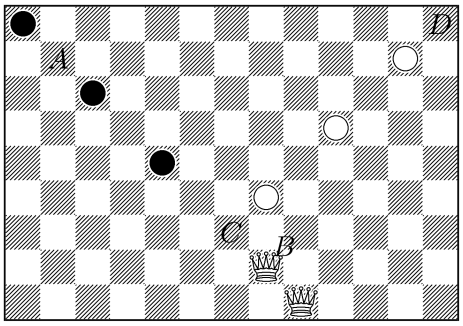
(a) Spomaľovač: ak čierny preskáče políčka A, biely musí brať dámou, čo je v našej pozícii strata času.



(b) Rozdeľovač: čierny prichádza z F na G; biely mu podhodí pešiaka na H alebo I, podľa toho, kam ho chce nasmerovať; čierny potom musí ďalej skákať ku J alebo K.



(c) Kríženie nepredstavuje žiaden problém: čierny prichádzajúci z A bude pokračovať smerom na a a z B na b.

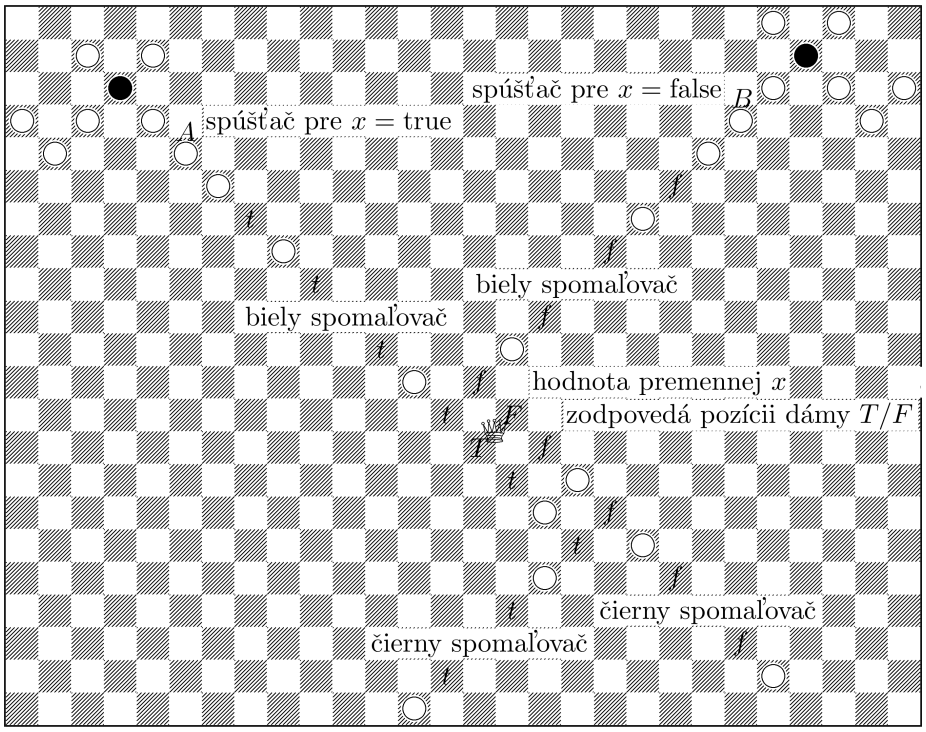


(d) Zmena farby: čierny signál prichádza z D až na C; tu prevezme iniciatívu biely a začne skákať dámou z B smerom na A.

Obr. 13.8: Gadgety redukcie G_3 na dámu.

zmenám ohodnotení premenných v hre G_3. Akonáhle je splnená niektorá klauzula čierneho, biely vyhrá tak, že aktivuje literály v danej klauzule. Ak na obr. 13.9 pohne pešiakom na pozícií A alebo B dopredu, odblokuje čierneho pešiaka nad ním, ktorý musí skákať po políčkach označených t (true), resp. f (false). Budeme hovoriť že biely *aktivoval literál* x*, resp.* $\overline{x}$.

Všimnite si, že čierny najskôr prejde cez biele spomaľovače a následne – ak je biela dáma na správnej pozícií – cez čierne spomaľovače; biely aj čierny v nich zabijú zopár ťahov, ale obaja rovnako veľa, takže hru to neovplyvní. Ak by však biely skúsil spustiť zlú vetvu – napr. dáma by bola na políčku F (false), ale pohol by pešiakom A, čierny pešiak by sa zasekol nad T a nevedel by ďalej pokračovať. Tým pádom by biely musel tráviť čas skákaním v spomaľovači a čierny by medzitým vyžral špirálu a vyhral by. Takže spustený signál musí naozaj zodpovedať nastaveniu premennej, inak hráč prehrá.



Obr. 13.9: Gadget pre bielu premennú. Dáma v strede sa presúva medzi pozíciami T a F, čo zodpovedá ohodnoteniu premennej true a false v hre G_3.

Takéto signály potom putujú od premenných až ku klauzulám, ktoré ich obsahujú ako v schéme na obr. 13.6. Keďže jedna premenná môže byť vo viacerých klauzulách, potrebujeme vedieť signál rozdeliť. Dosiahneme to ako na obr. 13.8b: čierny pešiak prichádzajúci z F doskáče na G, biely mu podhodí pešiaka buď na pozíciu H alebo I, podľa toho, kam chce signál ďalej viesť a čierny musí pokračovať. Cesty medzi premennými a klauzulami sa vo všeobecnosti môžu križovať, čo nie je problém (viď obr. 13.8c). Nakoniec tesne pred klauzulou vymeníme farby – pozri obr. 13.8d: doteraz pri aktivovaní bieleho literálu skákal čierny pešiak. Ten prichádza z D, priskáče až na C, odkiaľ začne biela dáma z B skákať smerom k A.

A tak sa signál o pravdivostnej hodnote nejakej premennej dostane až ku klauzule. Príklad čiernej klauzule s tromi bielymi a dvomi čiernymi premennými je znázornený na obr. 13.10. Bielym literálom zodpovedá signál zľava, cesty označené 1, 2 a 3 (cesta 3 je v dvoch kópiách, o chvíľu si povieme, prečo). Cesty 4 a 5 sprava zdola zodpovedajú *negáciam* čiernych literálov, tzn. ak napríklad klauzula obsahuje čierne literály y_1, $\overline{y_2}$, cesta bude viesť z vetvy pre y_1 = false a y_2 = true.

Pripomeňme, že akonáhle je po ťahu čierneho nejaká jeho klauzula splnená,

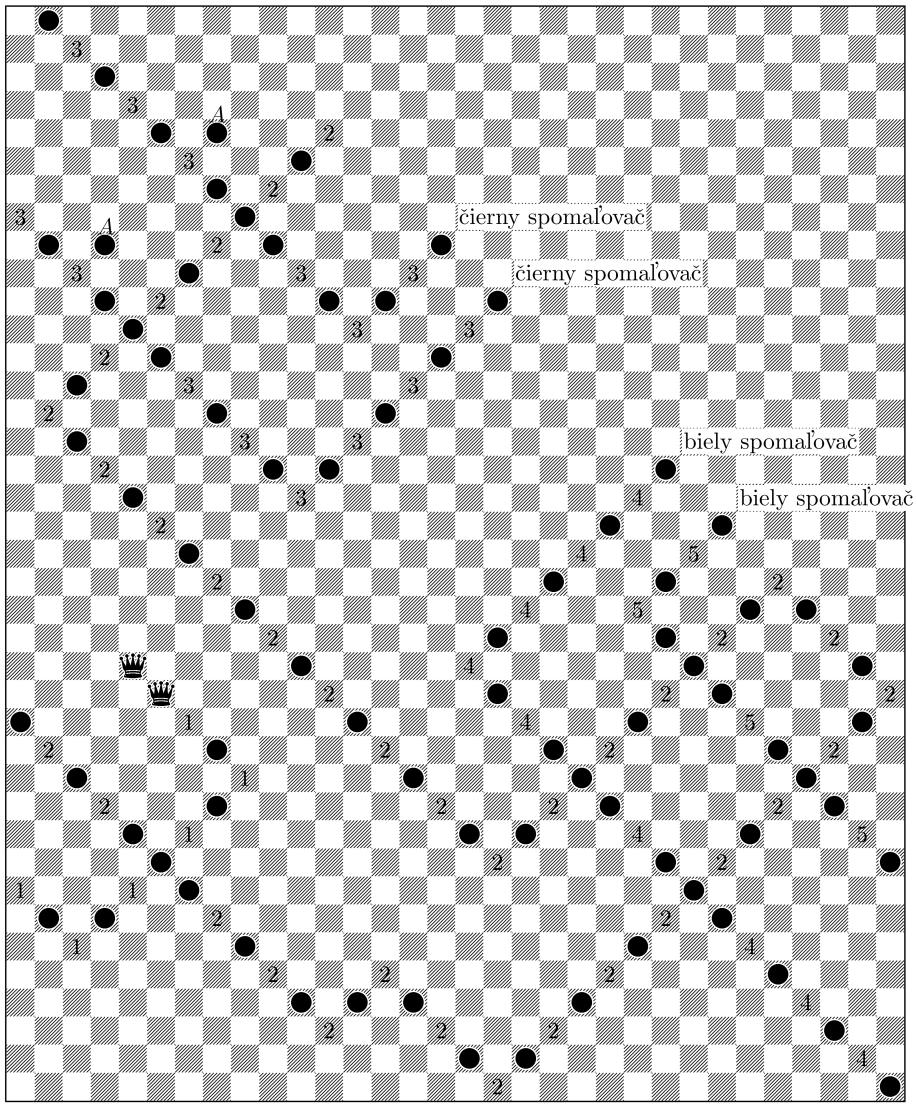
v hre G_3 prehral – a mal by teda prehrať aj v dáme. Biely vyhrá vtedy, keď aktivuje všetky svoje literály v splnenej klauzule a čierny nedokáže aktivovať žiadnu negáciu.

Všimnite si miesto, kde sa križujú cesty 1 a 2 – signál po druhej ceste môže pokračovať iba vtedy, keď sme ešte predtým zobrali čierneho pešiaka na križovatke a teda už sme vyzobali cestu číslo 1. Podobne po ceste číslo 3 môžeme prejsť len vtedy, ak sme už predtým prešli po 1 a 2 (v opačnom prípade zastaneme na križovatke a dámu vezme pešiak na pozícii A). Nakoniec však cesta 3 aktivuje čierny spomaľovač, takže ak majú všetky biele premenné správnu hodnotu, biely vyhrá.

A čo čierne premenné v klauzule? Všimnite si konce ciest 1 a 2. Keď biely aktivuje prvý literál, čierna dáma na konci cesty 1 musí brať späť, takže biely je opať na ťahu a aktivuje aj druhý literál, atď. Cesta číslo 2 (alebo vo všeobecnosti predposledný biely literál) však končí „do prázdna“ a čierny môže ťahať akokoľvek chce. V tomto momente má jeden (jediný) ťah k dobru a ak má niektorá čierna premenná zlú hodnotu, má šancu to dokázať. Každá biela klauzula je totiž napojená jednak na biele literály zľava a jednak na *negácie* čiernych literálov sprava. Ak teda biely aktivuje cesty 1 a 2 a negácia nejakého čierneho literálu je pravdivá, čierny na ťahu ho aktivuje a dostane sa k bielemu spomaľovaču ešte skôr ako stihne biely aktivovať svoj posledný literál (a čierny spomaľovač) – a teda vyhrá čierny.

Čo sa však stane, ak biela klauzula *je* naozaj pravdivá? Čierny má aj tak jeden ťah k dobru a môže spraviť *čokoľvek* chce. Napríklad môže prerušiť alebo zablokovať cestu 3. Práve preto vedú do každej klauzule dve kópie signálu z poslednej premennej – ak čierny jednu prekazí, použijeme tú druhú. Samotné premenné a rozdeľovače pokaziť nevie, ale čo ak poruší špirálu? Potom ju nebudeme vedieť jedným ťahom zobrať celú. Riešenie je také, že spomaľovače nám dajú dostatok času, aby sme postup z obr. 13.7 zopakovali na dvoch miestach prerušenú špirálu zobrali na dvakrát.

Posledný zádrhel: a čo ak jeden z hráčov nebude ťahať dámou medzi T a F, ale spraví ľubovoľný iný ťah? Napríklad preruší špirálu, klauzulu, alebo nejakú cestu k nej? Nuž v prvom rade, pre prípad, že hráč poškodí klauzulu, máme z každej klauzuly dve kópie (viď obr. 13.6). V druhom rade som na začiatku tak trochu klamal, že dôkaz je redukciou z G_3. V skutočnosti budeme vedieť simulovať iba tie hry G_3, ktoré vzniknú redukciou z hry G_1 a tie majú tú špeciálnu vlastnosť, že hráč na ťahu má vždy na výber práve dve premenné, z ktorých jednu musí zmeniť. A ak zmení inú premennú, alebo ak by nepotiahol, tak na konci ťahu splní nejakú klauzulu a prehrá. Nuž a ťah iný, ako je posunutie niektorej dámy medzi T a F, zodpovedá nepotiahnutiu v hre G_3, čo vedie k prehre. □



Obr. 13.10: Čierna klauzula pre 3 biele a 2 čierne literály, povedzme ($\overline{x_1} \wedge \overline{x_2} \wedge x_3 \wedge y_1 \wedge \overline{y_2}$). Cesty 1, 2, 3 zľava vedú z bielych premenných, cesty 4 a 5 sprava zdola zodpovedajú *negáciam* čiernych literálov. Ak chce biely vyhrať, musí aktivovať literály $\overline{x_1}$, $\overline{x_2}$, x_3 *a zároveň* čierny nemôže aktivovať negácie $\overline{y_1}$ a y_2.

13.4 Šach

Veta 13.5 (Fraenkel (1981)). *Šach je* EXP*-úplný.*

■ **Dôkaz.** Redukciou z G_3. Premenné simulujeme gadgetom ako na obr. 13.11 (gadget pre čiernu premennú dostaneme, ak obrázok preklopíme a vymeníme farby). Biela veža v stĺpci vpravo má prakticky dve možné pozície, ktoré predstavujú hodnoty true a false. Väčšinu hry budú hráči hýbať iba týmito vežami, čo zodpovedá ťahom v G_3. V momente, keď je nejaká súperova klauzula splnená, vyštartujú dámy z príslušných premenných a zaútočia. Čierna dáma útočí po jednej z červených ciest – horná zodpovedá $x =$ true, pretože je priechodná iba ak nie je strážená bielou vežou, t.j. veža je v pozícií true; podobne dolná cesta zodpovedá $x =$ false. Biela dáma zase útočí po jednej z modrých ciest – horná zodpovedá $x =$ true, dolná $x =$ false – cesta je priechodná, len keď ju vlastná veža neblokuje.

Predstavme si teraz, že (bez újmy na všeobecnosti) biela klauzula v G_3 je splnená a ukážme si, ako čierny dokáže vyhrať (opačný prípad je symetrický). Plán je nasledovný:

- Z premenných v klauzule (bielych aj čiernych) postupne vyštartujú čierne dámy. (3 ťahy)
- Prejdú cez „jednosmerku“ (obr. 13.12), ktorá zabezpečí, že už sa nevedia vrátiť – pri pokuse o návrat pohne biely pešiakom a odkryje modrú bodkovanú diagonálu stráženú strelcami. (5 ťahov)
- Čierny postupne nastaví všetky dámy do tunelu pre príslušnú klauzulu – pozri obr. 13.13: Pre každú premennú aj jej negáciu máme jeden zvislý tunel – tadiaľto prídu čierne dámy – a pre každú klauzulu máme diagonálny tunel. Plán útoku je, že čierny rozostaví dámy zo všetkých premenných v splnenej klauzule v príslušnom tuneli. Detaily križovatky sú znázornený na obr. 13.14a – tu čierna dáma vie zastať a obr. 13.14b – tu čierna dáma nemôže zastať, zobral by ju biely pešiak. Pozícia je zostrojená tak, že do tunela nejakej klauzuly sa môžu postaviť len dámy z literálov, ktoré patria do danej klauzuly.
- Späť k obr. 13.13. Keď čierny rozostaví dámy v tuneli pre klauzulu, odohrá sa rozhodujúci súboj o pešiaka na políčku s názvom „oltár“. Tento pešiak je strážený bielymi strelcami, ktorých je o 1 menej ako je literálov v klauzule, takže čierny obetuje svoje dámy a vyhrá práve vtedy, ak prišli všetky dámy, t.j. všetky premenné v klauzule mali správnu hodnotu a klauzula je splnená. Tento boj trvá podľa toho, koľko literálov je v klauzule – najviac 12.
- V prípade úspechu posledná čierna dáma pokračuje po červenej ceste. Poberie nechránených pešiakov, tzv. spomaľovače, ktorých jediný účel je vyrovnať počet ťahov tak, aby bol pre každú klauzulu rovnaký (keďže niektoré klauzuly môžu mať menej literálov).

- Posledné dva ťahy a čierny matuje ako na obr. 13.14c.

Celý tento útok, ak prepokladáme, že biely vymení na oltári všetkých strelcov, trvá presne

$$\underbrace{8}_{\text{dám}} \times \underbrace{8 \text{ ťahov}}_{\text{príchod do tunelov}} + \underbrace{7 \text{ ťahov}}_{\text{výmeny}} + \underbrace{3}_{\text{mat poslednou dámou}} = 74 \text{ ťahov.}$$

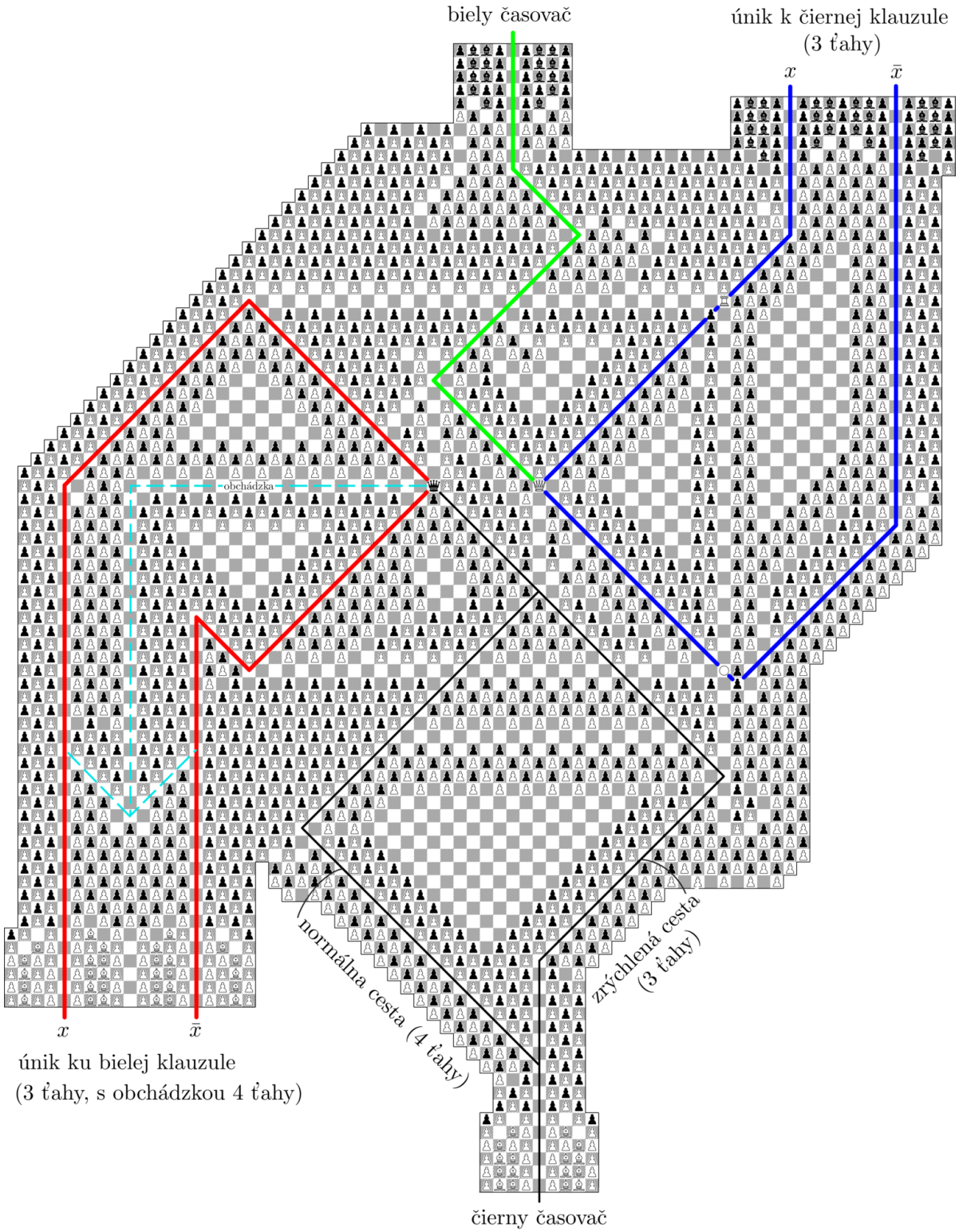
Ako sme spomínali, ak má klauzula menej ako 8 literálov, cestu doplníme spomaľovačmi tak, aby mal každý útok presne 74 ťahov.

Týmto sme popísali náš „preferovaný scenár“, ako by hra mohla prebiehať a ako jeden hráč dokáže vyhrať, ak je splnená súperova klauzula. Treba však ešte ukázať, čo sa stane, ak by hra prebiehala inak.

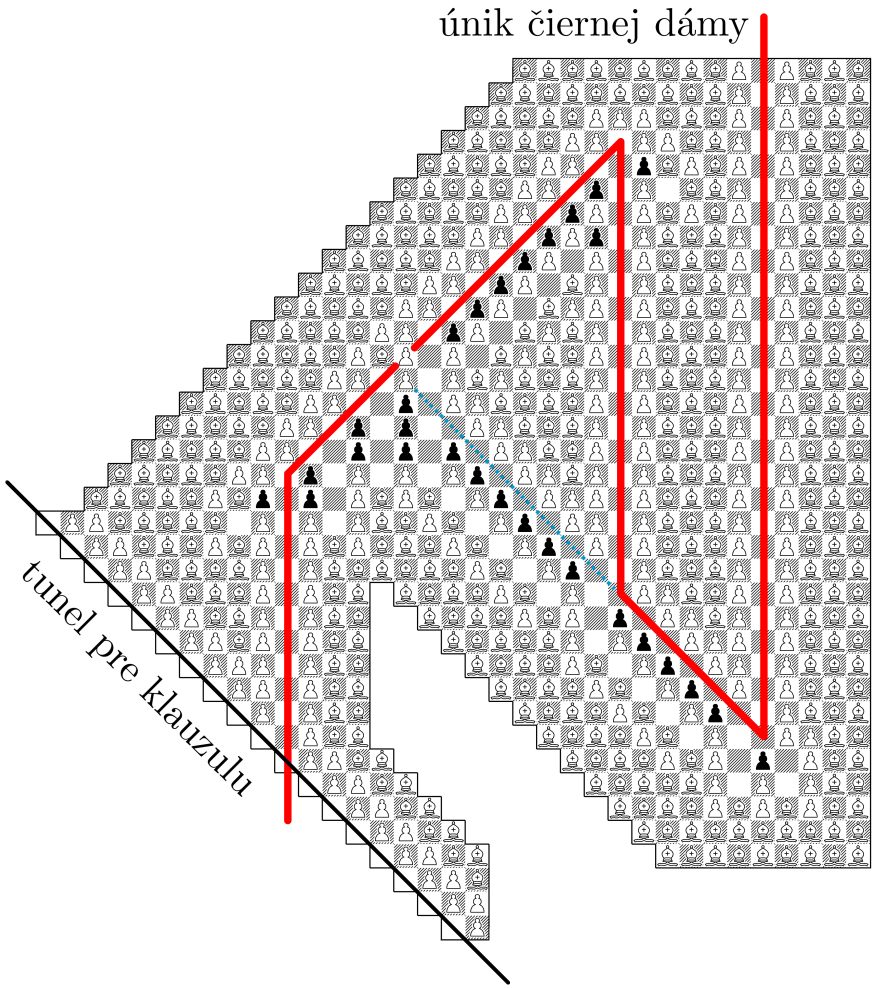
Napríklad, čo ak by sa čierny pokúsil niektoré dámy uvoľniť ešte počas prvej fázy hry, v nádeji, že zatiaľčo biely bude hýbať vežami a nastavovať biele premenné, čierny si uvoľní zopár dám, keď budú mať premenné zrovna vyhovujúcu hodnotu. V skutočnosti je takýto plán odsúdený k neúspechu. Akonáhle čierny opustí ľubovoľnou dámou jej pozíciu v strede gadgetu pre premennú, biely prejde do protiútoku a ak čierny nezmatuje do 74 ťahov, biely dá mat v 75-tom ťahu. Priebeh je nasledovný: Akonáhle čierna dáma opustí svoju pozíciu, biela dáma vyštartuje pozdĺž zelenej cesty (pozri obr. 13.11) zvanej „biely časovač“. Všimnite si, čierna dáma strážila zelenú cestu, takže sa nedala použiť skôr. Na konci tejto zelenej cesty je vhodné množstvo spomaľovačov (presne 62 pešiakov, ak dobre počítam; viď obr. 13.14c, ktorý však ukazuje čierny časovač) tak, že biela kráľovná matuje presne o jeden ťah neskôr, ako čierna (6 ťahov dámou + 62 spomaľovačov + 7 výmen bielych strelcov za čierne dámy na oltári = 75 ťahov). Takže v okamihu, keď jeden hráč zaútočí, je to buď-alebo – buď bola klauzula splnená a hráč vyhrá, alebo sa mu nepodarí preraziť a vyhrá súper. Preto sme tak starostlivo rátali počet ťahov. (Malé spresnenie: Útok nemusí trvať presne 74 ťahov, keďže biely nemusí vymeniť všetkých svojich strelcov, je to 67 + #výmen; biely môže namiesto výmen napr. pohnúť dámou v časovači, ale výsledok bude rovnaký, pretože biely spraví 68 + #výmen ťahov, teda matovanie mu trvá o 1 ťah viac ako čiernemu.)

Ďalšia možnosť, ktorá by sa mohla pritrafiť, je nasledovná: Čierny potrebuje niekoľko ťahov, kým unikne so všetkými dámami – čo keby biely medzičasom zmenil pozície bielych veží, teda zmenil ohodnotenie niektorých premenných? Nuž pre tento prípad gadget pre premennú obsahuje „obchádzku“ – na obr. 13.11 je táto cesta zobrazená prerušovanou tyrkysovou čiarou. Takže aj keď by sa biely vežou pokúšal zablokovať nejakú červenú cestu, čierna dáma môže uniknúť aj tak. Všimnite si však, že obchádzka trvá o jeden ťah viac, ako „priama“ červená cesta. To je v poriadku, pretože obchádzku použijeme len vtedy, keď biely spravil navyše ťah vežou, t.j. počet ťahov bude síce väčší, ale stále platí, že (ak je splnená biela klauzula) čierny matuje presne o jeden ťah skôr, ako biely.

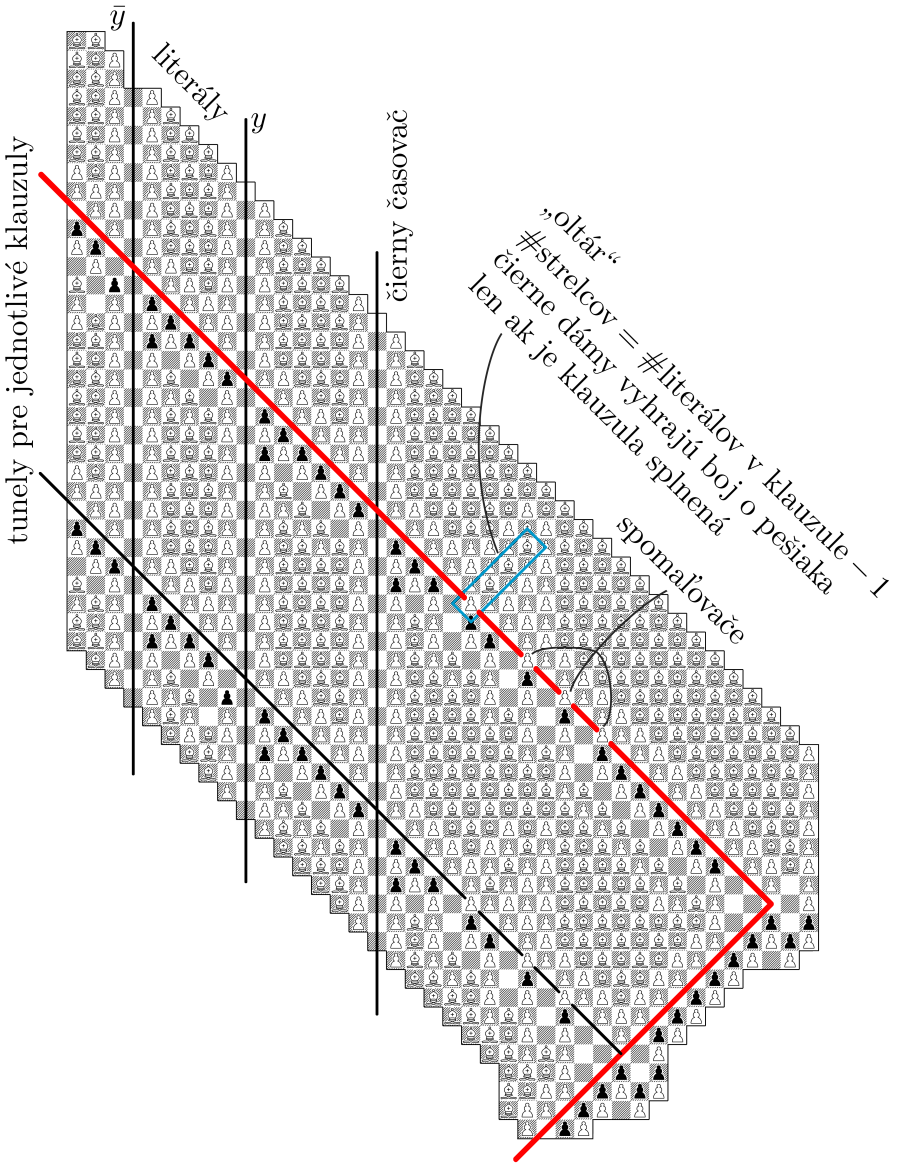
V poriadnom dôkaze treba premyslieť ešte všetky ostatné možnosti (čo ak sa veža pohne horizontálne? čo ak biely obetuje dámu? čo ak biely alebo čierny potiahne pešiakom? atď.) – my ich prenecháme čitateľovi. □



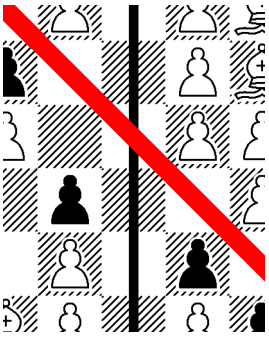
Obr. 13.11: Gadget pre bielu premennú. Pozícia veže v pravom stĺpci reprezentuje pravdivostnú hodnotu x – horná pozícia je false, dolná pozícia označená bielym krúžkom je true. Väčšinu hry sa budú veže premiestňovať medzi týmito pozíciami a strážiť únikovú cestu súperovej dámy. Ak je veža hore, horná cesta je strážená, ale čierna dáma môže na 3 ťahy uniknúť dolnou červenou cestou $\bar{x}$; ak zase stráži dolnú cestu, dáma môže uniknúť hornou červenou cestou x. Akonáhle je nejaká biela klauzula splnená, biele dámy vyrazia do útoku. Podobne biele dámy vyrazia do útoku po modrej ceste, akoháhle je nejaká čierna klauzula splnená. Tiež im stačia 3 ťahy – pokiaľ im vlastná veža nezavadzia. Keď čierna dáma vyrazí do útoku, biela dáma môže zahájiť protiútok po zelenej ceste nazývanej „časovač“ (a podobne ak biela dáma vyrazí do útoku, čierna začne protiútok cez čierny časovač).



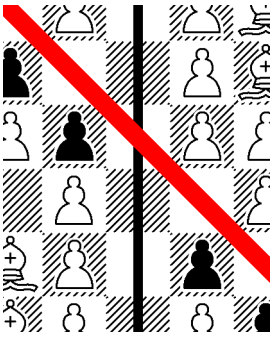
Obr. 13.12: Jednosmerka: Táto konštrukcia zabezpečuje, že čierna dáma môže prejsť po červenej ceste zhora nadol do tunela pre nejakú klauzulu, ale nemôže sa vrátiť. Po ceste totiž zoberie bieleho pešiaka a ak by sa pokúsila ísť naspäť, biely pohne pešiakom pod ním, čím uvoľní radu strelcov, ktorí ubránia modrú bodkovanú diagonálu. Rovnako dámy prislúchajúce iným premenným, ktoré sa dostanú do tunela sa nedokážu prebiť zdola.



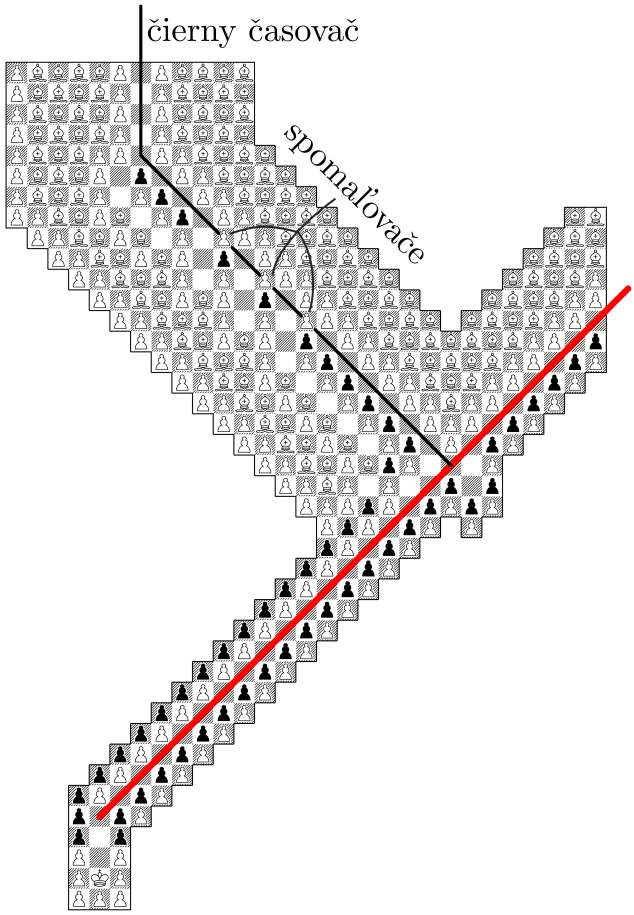
Obr. 13.13: Každej premennej aj jej negácii zodpovedá zvislý tunel a každej klauzule zodpovedá jeden tunel. Ak je niektorá klauzula W_{lose} splnená, čierne dámy vyštartujú z gadgetov pre príslušné premenné a zoradia sa v tuneli tejto klauzuly. Pešiaková štruktúra zabezpečuje, že dámy sa môžu zastaviť iba na tých miestach, kde klauzula obsahuje príslušný literál (pozri obr. **??**). Na to, aby mohol čierny matovať, potrebuje vyhrať kľúčový súboj o pešiaka na mieste zvanom „oltár“. Tento pešiak je chránený strelcami (o 1 menej ako je počet literálov v klauzule) a čierny tento súboj vyhrá, ak v tuneli zhromaždí dámy pre všetky literály, t.j. práve vtedy, keď je klauzula splnená. Ak je to tak, ostane mu jedna dáma, ktorá požerie spomaľovacie pešiaky, ktoré zabezpečujú, aby matovanie trvalo presný počet ťahov a pokračuje smerom nadol.



(a) Križovatka šikmého tunela pre klauzulu a zvislého tunela pre literál, ktorý patrí do klauzuly. Všimnite si, že čierna dáma sa môže postaviť na križovatku.



(b) Ak klauzula neobsahuje daný literál, pešiaková štruktúra je posunutá a čierna dáma nemôže prísť na túto križovatku – zobral by ju biely pešiak.



(c) Koniec hry: dáma, ktorá sa dostane cez oltár prichádza po červenej ceste a matuje bieleho kráľa. Druhá možnosť je, že dáma príde cestou, ktorú voláme „časovač“. Táto cesta trvá presne o 1 ťah dlhšie ako cesta cez oltár.

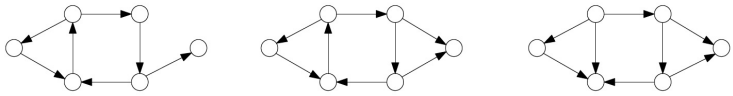
Obr. 13.14

Ďalšie EXP-úplné hry

- Medzinárodná dáma – pešiaci môžu brať aj dozadu; dáma sa pohybuje o ľubovoľný počet polí (Robson, 1984),
- Čínska dáma – hrací plán je šesťcípa hviezda, hráč sa pokúša dostať svoje figúrky do opačného cípu (Kasai a spol., 1979)
- Go (Robson, 1983)
- čínsky šach Xiangqi a kórejský šach Janggi (Zhang, 2019b),
- japonský šach Shogi (Adachi a spol., 1987),
- hra džungľa Dou Shou Qi (Zhang, 2019a),

Úlohy

- Na obrázku sú grafy pozícií troch rôznych hier. Ak hráč nemôže potiahnuť (z vrcholu nevedie žiadna hrana), prehráva. Zistite, ktoré pozície sú vyhrávajúce/prehrávajúce/remízové a aká je optimálna stratégia pre každú hru.



- Zafarbite zvyšné vyhrávajúce/prehrávajúce/remízové pozície na obr. 13.1.
- Ako by sa zmenila zložitosť šachu či dámy, ak by sme pridali podmienku, že po polynomiálnom počte ťahov bez výmeny/preskočenej figúrky hra končí remízou?

Literatúra

Adachi, Hiroyuki, Hiroyuki Kamekawa, a Shigeki Iwata. 1987. "Shogi on $n \times n$ board is complete in exponential time." *Trans. IEICE* 70, s. 1843–1852.

Fraenkel, David, Aviezri S a Lichtenstein. 1981. "Computing a perfect strategy for $n \times n$ chess requires time exponential in n." In *International Colloquium on Automata, Languages, and Programming*. Springer, s. 278–293.

Kasai, Takumi, Akeo Adachi, a Shigeki Iwata. 1979. "Classes of pebble games and complete problems." *SIAM Journal on Computing* 8(4), s. 574–586.

Robson, John Michael. 1983. "The complexity of Go." In *Proc. 9th World Computer Congress on Information Processing, 1983*. s. 413–417.

—. 1984. "N by N checkers is EXPTIME complete." *SIAM Journal on Computing* 13(2), s. 252–267.

Stockmeyer, Larry J a Ashok K Chandra. 1979. "Provably difficult combinatorial games." *SIAM Journal on Computing* 8(2), s. 151–174.

Zhang, Zhujun. 2019a. "A Note on Computational Complexity of Dou Shou Qi." *arXiv preprint arXiv:1904.13205* .

—. 2019b. "A note on hardness frameworks and computational complexity of Xiangqi and Janggi." *arXiv preprint arXiv:1904.00200* .

Záver

V tejto kapitole sme sa venovali hrám z pohľadu výpočtovej zložitosti. Ak vás táto téma zaujala a chcete sa dozvedieť viac, odporúčam knihu Hearn a Demaine (2009) a predmet Erika Demaina na MIT: *6.892 Algorithmic Lower Bounds: Fun with Hardness Proofs* s voľne dostupnými materiálmi a videami z prednášok.

Na záver sa poďme spolu ešte ohliadnuť späť za touto časťou. Skúsme si spraviť poriadok vo všetkých výsledkoch a charakterizovať jednotlivé hry. Ukazuje sa, že dôležité sú aspoň dve delenia: podľa počtu hráčov a podľa počtu ťahov, za ktoré hra skončí:

Hry pre jedného hráča (puzzle, solitairy, skákačky), ktoré končia v polynomiálnom počte ťahov sú v NP. Ak je dĺžka hry neobmedzená, hra sa dá vyriešiť v PSPACE = NPSPACE.

Ak už máme hru pre dvoch hráčov, ktorý hrajú proti sebe, striedanie ťahov zodpovedá alternácii. Hry, ktoré končia v polynomiálnom počte ťahov (reversi, piškvorky) sú v AP = PSPACE. Ak je dĺžka hry neobmedzená, hrá sa dá vyriešiť v APSPACE = EXP.

Ako sme videli v predchádzajúcich kapitolách, veľa hier je úplných vo svojej „prirodzenej“ kategórii.

	Hry pre 1 hráča	*Hry pre 2 hráčov*
polynomiálny #ťahov	NP/coNP-úplné (Tetris, Clickomania, míny, Sudoku, Mastermind, ...)	PSPACE-úplné (reversi, piškvorky, Hex, Amazonky, geografia,...)
neobmedzený #ťahov	PSPACE-úplné (Super Mario, Prince of Persia, Quake, Sokoban, Rush Hour,...)	EXP-úplné (dáma, šach, go, Shogi, Xiangqi,...)

Literatúra

Hearn, Robert A a Erik D Demaine. 2009. *Games, puzzles, and computation.* CRC Press.

Časť V

Logika, rozhodnuteľnosť a zložitosť

Úvod

V tejto časti budeme študovať rôzne logiky a ich problém rozhodnuteľnosti, t.j. dám vám logickú formulu a otázka znie, či je táto formula pravdivá.

Napríklad je pravda, že

$$\models \big((p \to q) \land (r \to s) \land (\neg q \lor \neg s)\big) \to (\neg p \lor \neg r)?$$

Platí v reálnych číslach

$$\mathbb{R} \models (\forall x)(\forall y)(\exists z)(x = y \cdot z)?$$

A platí v prirodzených číslach

$$\mathbb{N} \models (\forall x)(\exists y)(x < y) \land (\forall w)(\forall z)(y = w \cdot z \to w = 1 \lor z = 1)?$$

Alebo

$$\mathbb{N} \models (\forall x)(\exists y)(x < y) \land (\forall w)(\forall z) \quad ((y = w \cdot z \to w = 1 \lor w = 1) \\ \land (y + 2 = w \cdot z \to w = 1 \lor z = 1))?$$

Skúste si rozmyslieť, či sú tieto výroky pravdivé. Existuje nejaký spôsob, ako to mechanicky rozhodnúť? A ak áno, aký zložitý je tento problém?

V prvom prípade sa pýtame, či je pravda, že ak máme dve implikácie a vieme, že neplatí aspoň jeden z ich dôsledkov, tak neplatí aspoň jeden z predpokladov. Odpoveď je Áno (táto tautológia má názov „deštruktívna dilema“). Overiť si to môžeme tabuľkou a vyskúšaním všetkých 16-tich možností pre pravdivostné hodnoty p, q, r, s. Dá sa to aj lepšie? Vo všeobecnosti áno, trochu lepšie, ale nepoznáme žiaden polynomiálny algoritmus, čo má dosť dobrý dôvod: zistiť, či je formula výrokovej logiky tautológia, je coNP-úplný problém.

V druhom prípade sa pýtame, či pre každé x, y existuje výsledok $x/y = z$, teda, či sa dá každé číslo deliť každým. Odpoveď je samozrejme Nie: nulou sa nedá deliť, napríklad neplatí $\exists z : 1 = 0 \cdot z$. Vedeli by sme aj v takomto prípade mechanicky overiť, či je formula pravdivá? A ak áno, aká bude zložitosť takéhoto algoritmu?

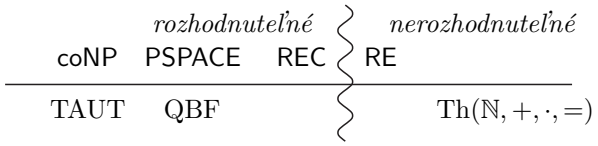
Tretí príklad v preklade hovorí, že existuje nekonečne veľa prvočísiel – to je pravda. No a štvrtý, že existuje nekonečne veľa prvočíselných dvojičiek (čísiel p takých, že p aj $p + 2$ sú prvočísla) – toto je otvorený problém, nikto nevie.

Známa Gödelova veta hovorí, že existujú tvrdenia o prirodzených číslach, ktoré sú pravdivé, ale nie sú dokázateľné (povedzme z Peanových axióm aritmetiky). My sa však na vec pozrieme z hľadiska informatiky a teórie zložitosti. Formuly sú „obyčajné“ reťazce symbolov nad abecedou $\{\wedge, \vee, \neg, \exists, \forall, =, (,), x, 0, 1, +, \cdot\}$. Jazyk dôkazov je rekurzívny (dôkaz vieme overiť v konečnom čase); jazyk dokázateľných tvrdení je rekurzívne vyčísliteľný (ak dôkaz existuje, vieme postupne skúšať všetky dôkazy, až kým ho nenájdeme). Dá sa však ukázať, že jazyk *pravdivých* tvrdení nie je ani len rekurzívne vyčísliteľný.

V nasledujúcej kapitole si ukážeme, že do čísiel vieme zakódovať celé postupnosti, resp. reťazce nad nejakou abecedou. Ukážeme si, ako zapísať formuly, ktoré hovoria niečo o týchto reťazcoch: napríklad, že reťazec kóduje výpočet daného Turingovho stroja. Zostrojíme formulu $\phi_{M,w}$, ktorá je pravdivá vtedy, ak stroj M na vstupe w zastane. Problém rozhodnúť, či je pravdivá nie je rekurzívny a rozhodnúť, či je pravdivá negácia nie je ani rekurzívne vyčísliteľný.

Vo zvyšku tejto časti sa budeme venovať rôznym teóriam, ktoré sú niekde na hranici rozhodnuteľného. Na jednej strane máme aritmetiku $(\mathbb{N}, +, \cdot, =)$, ktorá je nerozhodnuteľná, na druhej strane výrokovú logiku, ktorá je coNP-úplná, či predikátovú logiku, ktorá je PSPACE-úplná. Ako vyzerá svet medzi tým?

Napríklad, čo ak máme teóriu prirodzených čísel so sčítaním, ale bez násobenia? Bude táto teória stále nerozhodnuteľná? A čo ak sa budeme baviť o reálnych číslach a nie o prirodzených? A ak je skúmaná teória rozhodnuteľná, aké zložité je rozhodnúť, či je daná formula pravdivá? To sú otázky, ktoré budeme v študovať v nasledujúcich kapitolách.



Kapitola 14

Teória aritmetiky je nerozhodnuteľná

14.1 Nerozhodnuteľnosť aritmetiky

Pod teóriou aritmetiky, $\mathrm{Th}(\mathbb{N}, +, \cdot, =)$, máme na mysli *všetky pravdivé tvrdenia o prirodzených číslach*, pričom „tvrdenia“ sú formuly zložené z

- premenných $x, x', x'', \ldots$
- konštánt 0, 1
- logických spojok $\neg, \wedge, \vee$
- kvantifikátorov $\exists, \forall$
- rovnosti, sčítania a násobenia $=, +, \cdot$
- a zátvoriek.

Každá formula je reťazec nad abecedou $\{x, ', 0, 1, \neg, \wedge, \vee, \exists, \forall, =, +, \cdot, (,)\}$. Syntax tohto jazyka by sme vedeli zadefinovať veľmi exaktne, ale budeme sa venovať zaujímavejším veciam – aj tak by sme ju nedodržiavali. Kvôli čitateľnosti budeme formuly zapisovať voľne: budeme používať premenné $x, y, z, \ldots$, dvojbodku a zátvorky (rôzne typy) tak podľa citu a budeme používať skratky. Môžeme ich chápať ako „makrá“, ktoré stačí rozvinúť a dostaneme reťazec zložený len z povolených symbolov. Alebo sa môžeme tváriť, že nové symboly sú súčasťou jazyka (bohatšieho, ale rovnako silného).

Tak napríklad pomocou $\neg$ a $\wedge$ vieme „dodefinovať“ aj všetky ostatné logické spojky. Reláciu $x \leq y$ vieme definovať cez $=$ takto: $\exists \delta : x + \delta = y$ (pripomeňme, že pracujeme s prirodzenými číslami, takže $\exists \delta$ znamená $\exists \delta \in \mathbb{N}$, teda $\delta \geq 0$). Podobne vieme dodefinovať relácie ako $\neq$ a $<$.

Ak by sme nemali konštanty 0, 1, 2, vedeli by sme si ich aj tak dodefinovať, pretože nula je jediné číslo, pre ktoré $x + 0 = x$ $(\forall x)$ a jednotka jediné číslo, pre ktoré $x \cdot 1 = x$ $(\forall x)$. Mohli by sme našu formulu začať

$$\exists k_0 : \forall x : x + k_0 = x \wedge \exists k_1 : \forall x : x \cdot k_1 = x \wedge \exists k_2 : k_2 = k_1 + k_1 \wedge \cdots$$

s tým, že vo zvyšku formule by sme namiesto $0, 1, 2$, používali premenné k_0, k_1, k_2. Všetky ďalšie konštanty vieme vyskladať pomocou mocnín 2, sčítania a násobenia.

Podobne môžeme „definovať“ funkcie cez relácie. Napríklad $x \bmod y = r$ práve vtedy, keď

$$\exists q : x = yq + r \wedge r < y.$$

Funkcia mod síce nie je priamo súčasťou jazyka, ale môžeme ju chápať ako skratku, napríklad

$$\underbrace{(\underbrace{(ax + b) \bmod p}_{r_1}) \bmod m = 0}_{r_2}$$

vieme prepísať tak, že si medzivýsledky akoby „uložíme“ do pomocnej premennej, ktorú potom použijeme vo zvyšku výrazu:

$$\exists r_1 : (ax + b \bmod p = r_1 \wedge \exists r_2 : (r_1 \bmod m = r_2 \wedge (r_2 = 0))).$$

Následne môžeme rozpísať definíciu *relácie* „dačo mod dačo = dačo“:

$$\exists r_1 : \exists q_1 : ax+b = pq_1+r_1 \wedge r_1 < p \wedge \exists r_2 : \exists q_2 : r_1 = mq_2+r_2 \wedge r_2 < m \wedge r_2 = 0.$$

Ďalej vieme dodefinovať

- celočíselné delenie: $x/y = q$ práve vtedy, keď $\exists r : x = yq + r \wedge r < y$,
- reláciu „x je deliteľné d“, alebo skátene $d \setminus x$ práve vtedy, keď $x \bmod d = 0$.
- vlastnosť „x je prvočíslo“, $(x \in \mathbb{P})$: použijeme štandardnú definíciu – $x \geq 2$ a je deliteľné iba 1 a samým sebou:

$$x \geq 2 \wedge \forall d : d \setminus x \to (d = 1 \vee d = x).$$

- odčítanie: $x - y = z$ ak $z + y = x \vee (x < y \wedge z = 0)$.

Veta 14.1. *Pre daný Turingov stroj M a w vieme skonštruovať formulu $\phi_{M,w}$, ktorá je pravdivá práve vtedy, keď M na vstupe w zastane.*

Dôsledok 14.1 (nekonštruktívna Gödelova veta). *Majme ľubovoľnú „rozumnú“ axiomatizáciu aritmetiky – takú, kde v konečnom čase vieme skontrolovať, či je niečo axióma a či je niečo krok odvodenia (napríklad Peanove axiómy[1]). Ak je takáto teória korektná (všetko dokázateľné je pravdivé), potom je neúplná (obsahuje vetu, ktorá je pravdivá, ale nedokázateľná).*

[1] Peanove axiómy, PA, sú

■ **Dôkaz.** Jazyk všetkých dokázateľných formúl je rekurzívne vyčísliteľný (stačí postupne skúšať a kontrolovať všetky dôkazy, a akceptovať formulu, ak nájdeme jej dôkaz), zatiaľčo jazyk pravdivých formúl nie je: ak by sme totiž vedeli (aspoň čiastočne) rozhodovať pravdivosť formúl, vedeli by sme (aspoň čiastočne) rozhodovať komplement problému zastavenia. □

■ **Dôkaz vety.** Predpokladajme najskôr, že okrem sčítania a násobenia vieme ešte aj umocňovať – že pracujme v silnejšej teórii $\mathrm{Th}(\mathbb{N}, +, \cdot, \uparrow)$. Ako sa umocňovaniu vyhnúť, alebo ako ho dodefinovať si nechajme úplne na koniec.

Dôkaz je pomerne priamočiary, hoci technický: Hlavný problém je nájsť vhodný spôsob, ako reprezentovať rôzne údaje ako čísla tak, aby sme s nimi vedeli efektívne pracovať. Formulu $\phi_{M,w}$ potom zostrojíme podobne ako v dôkaze Cook-Levinovej vety.

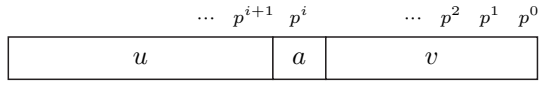
Predstavme si výpočet stroja M ako napríklad na obrázku 14.1a. Výpočet je *konečná* postupnosť konfigurácií – je to konečný reťazec a ten vieme zakódovať ako jedno obrovské číslo. Napríklad tak, že jednotlivé políčka zakódujeme ako cifry čísla (v sústave s dostatočne veľkým základom; pozri obrázok 14.1b).

Budeme pracovať v p-árnej sústave, kde p je dostatočne veľké prvočíslo[2]. Číslo x teda budeme chápať v tvare

$$x = x_{\ell-1}p^{\ell-1} + \cdots + x_i p^i + \cdots + x_2 p^2 + x_1 p^1 + x_0 p^0,$$

kde $x_i < p$ sú jednotlivé cifry.

Najzákladnejšia vec: Ako zistíme hodnotu i-tej cifry? Obrázkovo:



Uvedomíme si, že i-ta cifra je a práve vtedy, keď vieme x rozpísať nasledovne:

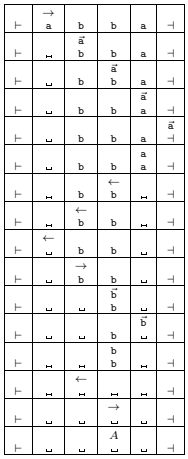
$$\begin{aligned} x &= \underbrace{(\cdots + x_{i+2}p + x_{i+1})}_{\substack{\text{horná časť}\\ \text{označme } u}} \cdot p^{i+1} + a \cdot p^i + \underbrace{(x_{i-1}p^{i-1} + \cdots + x_2 p^2 + x_1 p + x_0)}_{\substack{\text{spodná časť}\\ \text{označme } v}} \\ &= u \cdot p^{i+1} + a \cdot p^i + v = (u \cdot p + a) \cdot p^i + v, \end{aligned}$$

pričom cifra $a < p$ a spodná časť $v < p^i$. Napríklad druhá cifra odzadu v čísle 3141592 je 5, lebo v 10-tkovej sústave $3141592 = (3141 \cdot 10 + 5) \cdot 100 + 92$. Výraz „$x[i] = a$“ definujeme ako

$$\exists u, v : \; x = (u \cdot p + a) \cdot p^i + v \;\wedge\; a < p \;\wedge\; v < p^i.$$

- $\forall x : 0 \neq x + 1, \qquad \forall x, y : x + 1 = y + 1 \to x = y,$
- $\forall x : x + 0 = x, \qquad \forall x, y : x + (y + 1) = (x + y) + 1,$
- $\forall x : x \cdot 0 = 0, \qquad \forall x, y : x \cdot (y + 1) = x \cdot y + x,$
- $\forall \bar{y} : [\varphi(0, \bar{y}) \wedge \forall x : \varphi(x, \bar{y}) \to \varphi(x + 1, \bar{y})] \to \forall x : \varphi(x, \bar{y})$
  (schéma axióm indukcie; $\bar{y}$ je skratka pre $y_1, \ldots, y_k$)

[2] väčšie ako $(|\Gamma| + 2) \times (|Q| + 1)$



(a) Celý výpočet tvorí jeden reťazec, každé políčko konfigurácie je znak zo 45-prvkovej abecedy...

1	9	19	19	8	37
1	28	22	19	8	37
1	28	19	22	8	37
1	28	19	19	13	37
1	28	19	19	8	40
1	28	19	19	15	37
1	28	19	21	28	37
1	28	21	19	28	37
1	30	19	19	28	37
1	28	20	19	28	37
1	28	28	23	28	37
1	28	28	19	32	37
1	28	28	25	28	37
1	28	30	28	28	37
1	28	28	29	28	37
1	28	28	35	28	37

(b) ... môžeme ho zapísať aj ako 96-ciferné číslo v 47-čkovej sústave. Dohodnime sa, že toto číslo čítame odzadu, teda najvýznamnejšia cifra je vpravo dolu.

Obr. 14.1: Výpočet stroja, ktorý overuje, či je daný reťazec `abba` palindróm. Trvá 16 krokov a jedna konfigurácia má dĺžku 6. Na každom políčku je jeden z 5-tich symbolov (`a`, `b`, $-$, $\vdash$, $\dashv$) a buď tam nie je hlava, alebo tam je, v niektorom z 8-mich stavov ($\rightarrow$, $\leftarrow$, $\vec{\mathtt{a}}$, $\vec{\mathtt{b}}$, `a`, `b`, A, R). Výpočet teda môžeme zakódovať ako reťazec 16×6 znakov nad abecedou veľkosti 5×9.

Každá cifra môže kódovať jedno políčko ako dvojicu (stav, symbol). Stačí stavy očíslovať $1, \ldots, |Q|$, nula bude znamenať, že na políčku nie je hlava; a symboly (vrátane zarážok $\vdash\dashv$) $0, \ldots, |\Gamma| + 1$. Nech $m = |Q| + 1$; dvojicu (stav q, symbol a) potom môžeme reprezentovať cifrou

$$a \cdot m + q.$$

Naopak, môžeme dodefinovať funkcie

- stav na i-tom políčku je $Q[i] = x[i] \bmod m$ a
- symbol na i-tom políčku je $S[i] = x[i]/m$.

Zvyšok konštrukcie je podobný ako v Cook-Levinovej vete, kde sme pomocou premenných $Q^q_{i,j}, S^a_{i,j}$ vedeli popísať výpočet (s rozdielom, že pri SAT sme museli explicitne vypísať všetky klauzuly pre rôzne i, j; tu máme k dispozícii kvantifikátory).

Označme teda x číslo, ktoré kóduje celý výpočet a ℓ jeho dĺžku (počet cifier čísla x). Každý konečný výpočet zaberá len konečne veľa pamäte. Pre jednoduchosť doplňme na koniec voľné políčka „␣“ tak, aby každá konfigurácia mala rovnakú dĺžku k.

M na vstupe w zastaví, ak existuje konečný výpočet:

$$\phi_{M,w} \equiv \exists x : \textsc{Comp}_{M,w}(x).$$

Ostáva už „iba“ napísať formulu $\textsc{Comp}_{M,w}$, ktorá skontroluje, že x naozaj kóduje korektný výpočet stroja M, t.j., konfigurácie na seba nadväzujú a menia sa podľa príslušnej δ-funkcie, počiatočná konfigurácia má na vstupe slovo w a začína v počiatočnom stave a stroj sa nakoniec dostane do koncového (akceptačného alebo odmietacieho) stavu.

$$\begin{aligned}\textsc{Comp}_{M,w}(x) \equiv \exists p,k,\ell : k < \ell \wedge x < p^\ell \\ \wedge \textsc{Start}_{M,w}(x,k) \wedge \textsc{Valid}_M(x,k,\ell) \wedge \textsc{Halts}_M(x,\ell).\end{aligned}$$

- Vo formuli „$\textsc{Start}_{M,w}(x,k)$“ zadrátujeme počiatočnú konfiguráciu:

$$\begin{aligned}x[0] = \binom{}{\vdash} \wedge x[1] = \binom{q_0}{w_1} \wedge \cdots \wedge x[n] = \binom{}{w_n} \wedge n < k-1 \wedge x[k-1] = \binom{}{\dashv} \\ \wedge \forall i : (n < i \wedge i < k-1) \to x[i] = \binom{}{\sqcup}\end{aligned}$$

- Môžeme predpokladať, že predtým ako M zastane sa vždy vráti na ľavý okraj pásky a prejde do stavu A (ako *accept*) alebo R (ako *reject*). Či M zastavil, „$\textsc{Halts}_M(x,\ell)$“, potom overíme jednoducho:

$$\exists i : i < \ell \wedge (x[i] = \binom{A}{\vdash} \vee x[i] = \binom{R}{\vdash}).$$

- Kontrola, či konfigurácie správne nadväzujú podľa δ-funkcie: Pripomeňme, že k je dĺžka konfigurácie, takže políčku $x[i]$ v nasledujúcej konfigurácii zodpovedá políčko $x[i+k]$. Ak na pozícii i nie je hlava, tak tam ostane rovnaký symbol:

$$\forall i : Q[i] = 0 \to S[i] = S[i+k].$$

  Ak nie je hlava ani na susedných políčkach, tak tam nebude hlava:

$$\forall i : Q[i] = 0 \wedge Q[i-1] = 0 \wedge Q[i+1] = 0 \to Q[i+k] = 0.$$

  Nakoniec pre každé pravidlo δ-funkcie budeme mať formulu „ak je na políčku i hlava, ako sa zmenia políčka $i-1, i, i+1$ v nasledujúcej konfigurácii“. Napr. ak $\delta(p,a) = (q,b,-1)$, súčasťou formule bude

$$\forall i : x[i] = \binom{p}{a} \to (Q[i+k-1] = q \wedge x[i+k] = \binom{}{b} \wedge Q[i+k+1] = 0).$$

  Ak by sme trochu abstrahovali od detailov, mohli by sme označiť $\mathcal{C}$ množinu všetkých 6-tíc znakov (a,b,c,d,e,f) takých, že ak sú v nejakej konfigurácii tri po sebe idúce políčka (a,b,c) a v nasledujúcej konfigurácii, na tej istej pozícii je (d,e,f), je to konzistentné s δ-funkciou stroja M. Definujme

$$\textsc{Match}_M(x,i,k) \equiv \bigvee_{(a,b,c,d,e,f)\in\mathcal{C}} x[i]_3 = (a,b,c) \wedge x[i+k]_3 = (d,e,f),$$

kde $x[i]_3 = (a,b,c)$ je skratka pre $x[i] = a \wedge x[i+1] = b \wedge x[i+2] = c$. Potom

$$\textsc{Valid}_M(x,k,\ell) \equiv \forall i : (i+k+2 < \ell) \to \textsc{Match}_M(x,i,k).$$

A to je celá formula.

Vráťme sa teraz k otázke, či by sme to isté dokázali aj *bez* umocňovania. Asi tušíme, že odpoveď je áno – existuje viacero spôsobov, ako operáciu umocnenia dodefinovať. Ukážeme si však prekvapivé, vtipné riešenie: problém úplne obídeme a zaobídeme sa aj bez umocňovania.

Jedno miesto, kde ho potrebujeme je definícia $x[i] = a$ cez rozklad $x = (u \cdot p + a) \cdot p^i + v$. Finta je nasledovná: namiesto indexu i a počítania mocniny p^i budeme pracovať priamo s hodnotou $I = p^i$ a podobne namiesto k a ℓ budeme mať priamo hodnoty $K = p^k$ a $L = p^\ell$.

- Namiesto $i = 0, 1, 2, 3, \ldots$ budeme mať $I = p^0, p^1, p^2, p^3, \ldots$.
- Namiesto „pre všetky $i < \ell$“ budeme mať „pre všetky I, mocniny p menšie ako L“. Teda namiesto $(\forall i)$ budeme mať $\forall I : (I \text{ je mocnina } p) \to \cdots$.
- Podobne namiesto $(\exists i)$ budeme mať $\exists I : (I \text{ je mocnina } p) \wedge \cdots$.
- Namiesto $x[i] = a$ definujeme $x[\![I]\!] = a$:

$$I \text{ je mocnina } p \wedge \exists u, v : x = (u \cdot p + a) \cdot I + v \wedge a < p \wedge v < I.$$

- Namiesto $x[i+1]$ (vedľajšie políčko) budeme mať $x[\![I \cdot p]\!]$, totiž ak $I = p^i$, tak $I \cdot p = p^{i+1}$ je nasledujúca mocnina p.
- Namiesto $x[i+k]$ budeme mať $x[\![I \cdot K]\!]$, pričom predpokladáme, že $I = p^i$ a $K = p^k$, takže $I \cdot K = p^{i+k}$.

Nikde netreba počítať p^i – stačí mať hodnotu $I = p^i$ *už vypočítanú*. Jediné, čo potrebujeme, je vedieť *rozoznávať* mocniny p. A to je dôvod, prečo sme navrhovali *prvočíselnú* bázu p. Rozhodovať, či je nejaké číslo I mocnina p je totiž jednoduchšie pre prvočísla: je to vtedy, ak má I v prvočíselnom rozklade jediného prvočíselného deliteľa, a to p; inými slovami, ak p je jediné prvočíslo, ktoré delí I:

$$I \text{ je mocnina } p \quad \equiv \quad \forall d : (d \setminus I \wedge d \in \mathbb{P}) \to d = p. \qquad \square$$

Skúste si rozmyslieť, ako dôkaz upraviť a vytvoriť formulu $\phi_M(w)$, kde vstup nie je zadrátovaný ako vo $\phi_{M,w}$, ale je to parameter – voľná premenná.

Z doterajšieho pojednania by malo byť zrejmé, že číslami môžeme kódovať rôzne iné objekty, vrátane formúl či dôkazov. Vieme tiež zadefinovať predikát $\text{Dôkaz}_T(\pi, \psi)$, ktorý je pravdivý práve vtedy, keď π je korektný dôkaz tvrdenia ψ v teórii T. Jeden spôsob, ako to nahliadnuť je, že existuje Turingov stroj, ktorý skontroluje, či π je dôkaz ψ a ak áno, akceptuje. Skúste si rozmyslieť, ako by vyzeral priamy dôkaz.

Pre zaujímavosť, sú aj iné spôsoby ako túto vetu dokázať. Mocnina sa *dá* definovať. Konkrétne $a^b = c$, ak existuje postupnosť $m[0], m[1], m[2], \ldots, m[b]$ taká, že $m[0] = 1$ a vždy nasledujúci člen je a-krát väčší ($\forall i : m[i+1] = a \cdot m[i]$) – zjavne členy $m[i]$ sú potom mocniny a^i – a $a^b = c$, ak $m[b] = c$. Podobne funkciu faktoriál by sme vedeli definovať $n! = F$, ak existuje postupnosť $f[1], f[2], \ldots, f[n]$ taká, že $f[1] = 1$, $\forall i : f[i+1] = (i+1) \cdot f[i]$ a $f[n] = F$. Vo všeobecnosti vieme takýmto spôsobom, cez konečnú postupnosť pomocných medzivýsledkov, definovať všetky tzv. primitívne rekurzívne funkcie – *ak* ovšem vieme nejako kódovať konečné postupnosti.

Pôvodný Gödelov dôkaz kódoval konečné postupnosti cez zvyšky po delení. Dá sa dokázať, že pre každé n existuje[3] s také, že čísla $m_i = i \cdot s + 1$ sú nesúdeliteľné pre $i = 1, \ldots, n$. Podľa Čínskej zvyškovej vety potom pre ľubovoľné $x_1, \ldots, x_n$ existuje x (dokonca jedinečné x mod $M = \prod_i m_i$) také, že

$$\begin{aligned} x_1 &\equiv x \pmod{m_1} \\ x_2 &\equiv x \pmod{m_2} \\ &\vdots \\ x_n &\equiv x \pmod{m_n}. \end{aligned}$$

Inými slovami na $x \pmod M$ sa môžeme ekvivalentne pozerať ako na n-ticu prvkov modulo m_i. Algebraik by povedal, že Čínska zvyšková veta definuje izomorfizmus $\mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_n} \cong \mathbb{Z}_M$.

Gödel potom používal ďalšie kódovanie pomocou exponentov v prvočíselnom rozklade, napríklad postupnosť $3, 1, 4, 1, 5, 9, 2$ zakódoval ako jedno veľké číslo:

$$518251703594704429350000 = 2^{\mathbf{3}} \cdot 3^{\mathbf{1}} \cdot 5^{\mathbf{4}} \cdot 7^{\mathbf{1}} \cdot 11^{\mathbf{5}} \cdot 13^{\mathbf{9}} \cdot 17^{\mathbf{2}}.$$

Vo všeobecnosti: postupnosť $x_1, x_2, \ldots, x_n$ zakódoval ako číslo $p_1^{x_1} \cdot p_2^{x_2} \cdot \cdots \cdot p_n^{x_n}$, kde p_k je k-te prvočíslo. Skúste si rozmyslieť, ako zapísať výraz „x je k-te prvočíslo“ a „x má v prvočíselnom rozklade p^k“, teda „k je najvyššia mocnina p, ktorá delí x“.

Mimochodom, každý exponent je opäť prirodzené číslo a to môže kódovať konečné postupnosti. Jedným číslom teda môžeme kódovať aj postupnosť postupností prirodzených čísel. Gödel takto kódoval dôkaz ako postupnosť formúl, pričom každá formula bola postupnosť znakov. Podobne by sme mohli kódovať výpočet ako postupnosť konfigurácii, kde každá konfigurácia je postupnosť znakov. Aj pri našom kódovaní sme mohli zobrať dve prvočísla p_1, p_2, pričom cifry čísla (pri základe p_1) by boli jednotlivé konfigurácie a každá cifra by bola jedno veľké číslo, ktorého cifry (pri základe p_2) by boli jednotlivé políčka. Avšak, ako vieme z programovania, dvojrozmerné polia sa dajú simulovať jednorozmernými.

14.2 Gödelove vety

V predchádzajúcej kapitole sme si ukázali trochu slabšiu verziu Gödelovej vety – jednak sme ukázali iba to, že nejaká nedokázateľná veta existuje, neukázali

[3] stačí zobrať napríklad $s = k!$ pre $k > n$

sme žiadnu konkrétnu takúto vetu, jednak sme predpokladali, že naša teória je korektná, hoci stačí, aby bola bezosporná (pre každú formulu ψ sa dá dokázať najviac jedna z dvojice ψ a $\neg\psi$). Dôkaz je podobný ako ten, že problém zastavenia nie je rekurzívny:

Uvažujme nasledovný program Popleť. Ten na vstupe dostane kód nejakého Turingovho stroja M a začne hľadať dôkaz π tvrdenia $\phi_M(\langle M\rangle)$ alebo tvrdenia $\neg\phi_M(\langle M\rangle)$. Pripomeňme, že dôkazy sú len konečné reťazce, takže ich vieme postupne enumerovať a ak sa v teórii aritmetiky dá dokázať, že M na vstupe $\langle M\rangle$ zastane, alebo že sa zacyklí, tak ten dôkaz v konečnom čase nájdeme. Náš program Popleť však spraví nasledovnú habaďúru: ak nájde dôkaz, že $M(\langle M\rangle)$ zastane, tak sa zacyklí a ak nájde dôkaz, že $M(\langle M\rangle)$ sa zacyklí, tak zastane.

Vytvorme teraz program Naopak = Popleť($\langle$Popleť$\rangle$), teda Naopak iba spustí Popleť na svojom vlastnom kóde. Otázka za milión: ako sa správa Naopak? Zastaví? Zacyklí sa?

Všimnime si, že Popleť skúma, ako sa správa stroj M na svojom vlastnom kóde. Keď teda programu Popleť podhodíme jeho vlastný kód, bude skúmať, ako sa správa na vlastnom kóde – dostali sme autoreferenciu – program Naopak bude skúmať sám seba! Ak nájde dôkaz, že zastane, tak sa zacyklí a ak nájde dôkaz, že sa zacyklí, tak zastane, čo je spor.

Nemôže byť preto dokázateľná

$$\text{ani veta } \Gamma \equiv \phi_{\text{Popleť},\langle\text{Popleť}\rangle} \text{ ani jej negácia } \neg\phi_{\text{Popleť},\langle\text{Popleť}\rangle}.$$

Existencia dôkazu by znamenala, že program spraví opak a bolo by dokázateľné nepravdivé tvrdenie.

Ba čo viac: Ak hocijaký program zastane, tak to vieme dokázať a dokonca v teórii aritmetiky, napríklad z Peanových axióm – existuje totiž konkrétne číslo, ktoré kóduje tento konečný výpočet a stačí overiť, že toto číslo spĺňa všetky podmienky. To znamená, že keby program Naopak zastal, tak v aritmetike vieme dokázať vetu Γ. Nech π je lexikograficky najmenší, prvý dôkaz tvrdenia Γ alebo negácie $\neg\Gamma$. Ak π je dôkaz tvrdenia $\neg\Gamma$, tak máme sporný systém. Ak je však π dôkaz Γ, tak by sme mali vedieť dokázať, že po konečnom počte krokov Naopak preverí všetky predchádzajúce dôkazy, tie nevyhovujú a príde až k π, overí ho a presvedčí sa, že je to korektný dôkaz a zacyklí sa. Tento celý argument (že existuje nejaký konečný výpočet, ktorý sa dostane až do bodu, kde sa program zacyklí) sa dá sformalizovať ako dôkaz v teórii aritmetiky a opäť dostávame, že existuje dôkaz pre $\neg\Gamma$. Takže ak program zastane, naša teória je sporná. Ak predpokladáme, že sporná nie je, potom Naopak nezastane, zároveň sa to však nedá dokázať.

Veta 14.2 (Gödelova veta o neúplnosti). *Majme ľubovoľnú rozumnú axiomatizáciu aritmetiky (napríklad Peanove axiómy). Ak je takáto teória bezosporná, tak je neúplná.*

■ **Dôkaz.** Veta $\neg\Gamma$ je pravdivá, ale nedokázateľná. □

Veta 14.3 (Druhá Gödelova veta o neúplnosti). *Ľubovoľná dostatočne silná teória obsahujúca aritmetiku (napríklad Peanove axiómy) nevie dokázať vlastnú bezospornosť.*

■ **Náznak dôkazu.** Bezospornosť nejakej teórie sa dá zapísať napríklad ako $\mathrm{BSP}_T \equiv \nexists \pi : \mathrm{Dôkaz}_T(\pi, \langle 0 = 1 \rangle)$. V odstavci „Ba čo viac:“ vyššie máme *slovný* dôkaz, že ak je teória bezosporná, tak veta $\neg\Gamma$ je pravdivá. V dostatočne silnej teórii sa tento slovný dôkaz dá prepísať do formálneho dôkazu tvrdenia $\mathrm{BSP}_T \to \neg\Gamma$. Ak by sa v T dala dokázať jej vlastná bezospornosť, spojením týchto dôkazov by sme dostali dôkaz $\neg\Gamma$ a to je spor. □

14.3 Aritmetická hierarchia

Na záver si ešte neodpustíme pár slov o aritmetickej hierarchii (AH), ktorá bola inšpiráciou pre polynomiálnu hierarchiu (PH), ktorú sme študovali v kapitole 6. Jediný rozdiel je ten, že budeme hovoriť o *konečnom* čase (nie polynomiálnom). Teda namiesto triedy P budeme uvažovať rekurzívne jazyky REC, namiesto polynomiálnej redukcie budeme mať rekurzívnu redukciu (v konečnom čase).

Spodok hierarchie tvoria rekurzívne jazyky $\Delta_1 = \mathsf{REC}$, rekurzívne vyčísliteľné jazyky $\Sigma_1 = \mathsf{RE}$ a ich komplementy $\Pi_1 = \mathsf{coRE}$. Vo všeobecnosti $\Pi_n = \mathsf{co}\Sigma_n$.

AH môžeme definovať cez rekurzívne predikáty:

- Σ_k je trieda jazykov, pre ktoré existuje rekurzívna relácia R taká, že $x \in L \iff \exists y_1 \forall y_2 \exists \cdots Q y_k (x, y_1, \ldots, y_k) \in R$
- Π_k je trieda jazykov, pre ktoré existuje rekurzívna relácia R taká, že $x \in L \iff \forall y_1 \exists y_2 \forall \cdots Q y_k (x, y_1, \ldots, y_k) \in R$

... alebo cez orákulá:

$$\begin{array}{ccccccccc} \mathsf{REC} & \subseteq & \mathsf{RE} & \subseteq & \mathsf{RE}^{\mathsf{RE}} & \subseteq & \mathsf{RE}^{\mathsf{RE}^{\mathsf{RE}}} & \subseteq & \mathsf{RE}^{\mathsf{RE}^{\mathsf{RE}^{\mathsf{RE}}}} \subseteq \cdots \\ \| & & \| & & \| & & \| & & \| \\ \Sigma_0 & & \Sigma_1 & & \Sigma_2 & & \Sigma_3 & & \Sigma_4 \end{array}$$

Vo všeobecnosti $\Delta_{n+1} = \mathsf{REC}^{\Sigma_n}$, $\Sigma_{n+1} = \mathsf{RE}^{\Sigma_n}$ a $\Pi_{n+1} = \mathsf{coRE}^{\Sigma_n}$.

Rôzne úrovne majú svoje prirodzené úplné problémy. Napríklad

- problém zastavenia je Σ_1-úplný,
- problém prázdnosti (pre daný stroj M, je $L(M) = \emptyset$?) je Π_1-úplný,
- problém totálnosti (zastane M na každom vstupe?) je Π_2-úplný,
- problém finítnosti (je množina $L(M)$ konečná?) je Σ_2-úplný,
- problém kofinítnosti (je doplnok $\overline{L(M)}$ konečný?) je Σ_3-úplný.

Na rozdiel od polynomiálnej hierarchie vieme o tej aritmetickej dokázať viac:

1. $\Delta_n = \Sigma_n \cap \Pi_n$ (teda platí rovnosť, napríklad $\mathsf{REC} = \mathsf{RE} \cap \mathsf{coRE}$; v PH vieme dokázať iba inklúziu, napríklad $\mathsf{P} \subseteq \mathsf{NP} \cap \mathsf{coNP}$, ale nevieme, či sú tieto triedy rovnaké),

2. AH je striktná – vieme dokázať, že $\Sigma_k \subset \Delta_{k+1} \subset \Sigma_{k+1}$; napríklad spomínaný problém zastavenia patrí do RE, ale nie do REC; keďže hierarchia je striktná, problémy úplné pre nejakú úroveň hierarchie nemôžu patriť o úroveň nižšie

Viac sa čitateľ dozvie na teórii vypočítateľnosti. Túto kapitolu zakončime ešte uvedomením, že tak, ako sme dokázali zostrojiť formulu pre problém zastavenia, vieme zostrojiť formulu pre ľubovoľný problém z AH: vieme totiž definovať všetky rekurzívne predikáty (existuje akceptačný výpočet) a každý problém v AH sa dá zapísať pomocou takéhoto predikátu a niekoľkých kvantifikátorov. Naopak, problém rozhodnúť, či je daná formula pravdivá patrí do AH – na ktorú úroveň, záleží od počtu kvantifikátorov – takže rozhodovanie pravdivosti pre aritmetické problémy je rovnako ťažké, ako celá aritmetická hierarchia.

Existujú aj „ťažšie“ problémy? Ale pravdaže. Stále platí, že formúl je len spočítateľne nekonečne veľa, zatiaľčo všetkých problémov (jazykov) je nespočítateľne nekonečne veľa. Ešte ťažšie problémy dostaneme, ak povolíme kvantifikáciu nielen cez čísla, ale aj cez funkcie a relácie. Takáto logika druhého rádu potom definuje tzv. analytickú hierarchiu, ktorá obsahuje celú AH a ešte aj problémy oveľa ťažšie. Pojednanie o nich je už však nad rámec tohto textu.

Úlohy

- Dokážte, že teória $\mathrm{Th}(\mathbb{N}, <, \cdot)$ je nerozhodnuteľná. (Dokážte, že pomocou $<$ a $\cdot$ sa dá dodefinovať funkcia nasledovník $s : x \mapsto x + 1$ a všeobecne sčítanie.)

- Dokážte, že $\mathrm{Th}(\mathbb{Z}, +, \cdot)$ je nerozhodnuteľná. (Ako dodefinujeme reláciu $\leq$?)

Literatúra

Kapitola 15

Zložitosť teórií sčítania

V tejto kapitole sa pozrieme na dve slabšie teórie:

- tzv. Presburgerovu aritmetiku $\mathrm{Th}(\mathbb{N}, +, =, 0, 1)$ a
- teóriu reálnych čísel so sčítaním $\mathrm{Th}(\mathbb{R}, +, \leq, 0, 1)$.

Ukážeme, že problém rozhodnuteľnosti je v oboch prípadoch riešiteľný, ale

- pre $\mathrm{Th}(\mathbb{R}, +, \leq)$ patrí niekde medzi NEXP a $\mathsf{EXPSPACE}$ a
- Presburgerova aritmetika $\mathrm{Th}(\mathbb{N}, +, =)$ je niekde medzi $\mathsf{2NEXP}$ a $\mathsf{2EXPSPACE}$ (dvojito-exponenciálny čas a priestor).

To „niekde medzi“ sa dá povedať aj presnejšie. Pripomeňme definíciu triedy $\mathsf{STA}(s(n), t(n), a(n))$ jazykov, ktoré sa dajú rozoznať v priestore $O(s(n))$, čase $O(t(n))$, s pomocou $a(n)$ alternácií. Pripomeňme tiež, alternujúci čas sa zhruba rovná priestoru, napríklad $\mathsf{AEXP} = \mathsf{EXPSPACE}$ a $\mathsf{2AEXP} = \mathsf{2EXPSPACE}$. Spomínané problémy rozhodnuteľnosti sú úplné pre triedy s lineárne veľa alternáciami, čo je naozaj niekde medzi nedeterminizmom (len jedna alternácia) a neobmedzenou alternáciou. Konkrétne teória reálnych čísel je $\mathsf{STA}(*, 2^{O(n)}, n)$-úplná a Presburgerova aritmetika $\mathsf{STA}(*, 2^{2^{O(n)}}, n)$-úplná.

15.1 Horné odhady

Teória usporiadania

Skôr ako sa pustíme do reálnych čísel so sčítaním, ako rozcvičku uvažujme teóriu $\mathrm{Th}(\mathbb{R}, \leq)$. Takáto teória sa dá popísať axiómami pre tzv. husté lineárne usporiadanie bez koncových bodov:

- rovnosť: relácia $=$ je reflexívna, symetrická a tranzitívna,
- čiastočné usporiadanie: relácia $\leq$ je reflexívna, antisymetrická a tranzitívna,

- linearita: pre každé x, y je $x \leq y$ alebo $y \leq x$,
- hustota: pre každé $x < z$ existuje y striktne medzi nimi: $x < y < z$,
- neexistencia koncových bodov: pre každé x existuje ostro menší a ostro väčší prvok.

Rovnaké axiómy spĺňa $\mathrm{Th}(\mathbb{Q}, \leq)$.

Je táto teória rozhodnuteľná? A aká je zložitosť?

Veta 15.1. *Problém rozhodnúť, či je tvrdenie z* $\mathrm{Th}(\mathbb{R}, \leq)$ *pravdivé, je* PSPACE*-úplný.*

■ **Dôkaz.** Ťažkosť: Redukciou z QBF. Formulu $Q_1 x_1 \cdots Q_n x_n \phi(\vec{x})$ zakódujeme tak, že každú booleovskú premennú x_i nahradíme dvojicou reálnych premenných x_i, y_i, pričom $x_i \leq y_i$ bude znamenať dajme tomu TRUE a $x_i > y_i$ FALSE. Napríklad

$$\forall x_1 \exists x_2 : (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2)$$
$$\downarrow$$
$$\forall x_1, y_1 \exists x_2, y_2 : (x_1 \leq y_1 \vee x_2 > y_2) \wedge (x_1 > y_1 \vee x_2 \leq y_2).$$

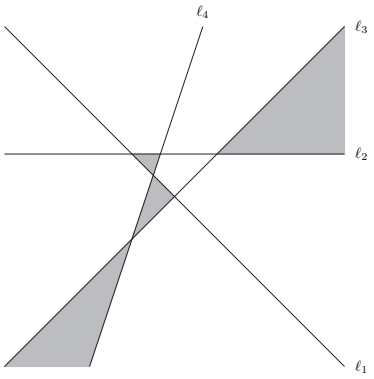
Vo všeobecnosti, hovoríme, že štruktúra je netriviálna, ak má aspoň jednu reláciu R takú, že $R(\vec{a})$ a $\neg R(\vec{b})$ pre nejaké k-tice $\vec{a}, \vec{b}$. Rozhodovací problém je pre netriviálne štruktúry aspoň PSPACE-ťažký.

PSPACE algoritmus: Na malú chvíľu sa sprvoti možno človek zháči, že keď pracujeme s reálnymi číslami, budeme musieť skúšať a overovať nekonečne veľa hodnôt. Ale nie je to tak. Predstavme si, že formula začína napríklad $\forall x \exists y \forall z \cdots : \varphi(x, y, z, \ldots)$. Za x stačí zvoliť jedinú možnosť: $x = 0$. Prečo? Nuž pre hocakú inú možnosť $a \in \mathbb{R}$ dostaneme rovnaký výsledok, pretože ak všetky hodnoty posunieme o a, hodnota φ sa nezmení. Poďme teraz vybrať y. Tvrdím, že stačí vybrať tri hodnoty, napríklad: $0, +1$ a -1. Prečo? Pre ľubovoľnú inú voľbu $b \neq 0$ by sme mohli všetky hodnoty predeliť $|b|$ a hodnota φ by sa nezmenila. Jednoducho φ rozlišuje len vzájomnú polohu bodov (či sú vľavo/vpravo), nič iné (nezáleží napríklad na vzdialenosti bodov). Intuitívne nemusíme skúšať všetky $x, y, z, \ldots \in \mathbb{R}$; stačí všetky neostré lineárne usporiadania. Ak už máme vybraté nejaké body, pre každú ďalšiu premennú stačí vyskúšať jeden z nich, číslo menšie alebo väčšie ako všetky ostatné a číslo medzi každou dvojicou.

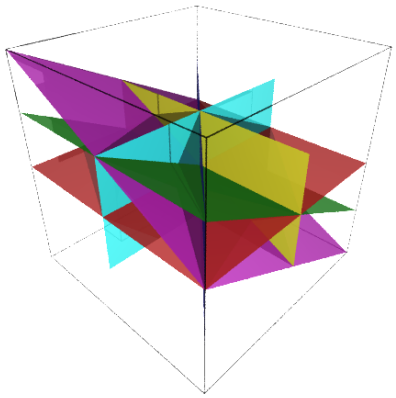
Trochu formálnejšie: majme formulu $Q_1 x_1 \cdots Q_n x_n \phi(\vec{x})$. Definujme reláciu ekvivalencie na n-ticiach $\vec{a} \equiv \vec{b}$, ak vzájmné poradie členov a_i je rovnaké ako vzájomné poradie b_i. Potom platí:

$$\vec{a} \equiv \vec{b} \qquad \Longrightarrow \qquad \models \phi(\vec{a}) \iff \models \phi(\vec{b})$$

Nie je problém v polynomiálnom priestore, resp. s lineárnym počtom alternácií prezrieť všetky možné vzájomné poradia (pre každé i, j máme $x_i < x_j$, $x_i = x_j$, alebo $x_i > x_j$). □



(a) Priamky $\ell_1, \ldots, \ell_4$ vo ϕ $(*)$ rozdelia rovinu na 11 oblastí. Šedou farbou sú vyznačené body, kde je $\phi(x, y)$ pravdivá.



(b) Rozdelenie priestoru v 3D. Afínne funkcie sú roviny, ktoré rozdelia $\mathbb{R}^3$ na niekoľko konvexných mnohostenov.

Obr. 15.1

Teória reálneho sčítania

Geometria

V teórii bez násobenia sa atomické formuly skladajú iba zo súčtu premenných a konštánt. Zaveďme skratku

$$n \times x = \underbrace{x + x + \cdots + x}_{n}.$$

Keďže z rovnosti $p \times x = q \times y$ vyplýva, že $y = (p/q) \times x$ a v ďalších výrazoch môžeme používať y, môžeme rovno povoliť racionálne konštanty. V takom prípade môžeme písať atomické formule (po prehodení na jednu stranu) v tvare

$$\sum_i a_i \times x_i \gtreqless b \qquad (a_i, b \in \mathbb{Q}),$$

čo sú lineárne (ne)rovnice (odtiaľ tiež názov „lineárna reálna aritmetika“).

Vezmime si napríklad formulu $\forall x : \exists y : \phi(x, y)$, kde

$$\begin{aligned}\phi(x,y) = &\Big[\big((y - 3x > 3) \vee (y > 2)\big) \wedge (y - x < 1)\Big] \\ &\vee \Big[(y < 2) \wedge (y - x > 1) \wedge \big((x + y > 1) \leftrightarrow (y - 3x > 3)\big)\Big] \qquad (*)\end{aligned}$$

Je táto formula pravdivá? V dvoch rozmeroch si môžeme nakresliť obrázok a – pozriem a vidím. Nerovnosti vo ϕ sú definované priamkami

$$\begin{aligned} y &= -x + 1 && (\ell_1) \\ y &= 2 && (\ell_2) \\ y &= x + 1 && (\ell_3) \\ y &= 3x + 3 && (\ell_4) \end{aligned}$$

Ak si ich narysujeme a vyznačíme oblasti, kde je formula $\phi(x, y)$ pravdivá, dostaneme obrázok 15.1a. Z neho vidno, že $\forall x : \exists y : \phi(x, y)$ je nepravdivá, ale $\forall y : \exists x : \phi(x, y)$ je pravdivá.

Na obrázku 15.1b je znázornený príklad v troch rozmeroch (s tromi premennými). Vo všeobecnosti lineárne funkcie definujú roviny, ktoré priestor rozdelia na konečne veľa oblastí, kde sa pravdivostná hodnota formule nemení. Stačí teda vyskúšať jeden bod z každej oblasti, plus samozrejme body na hraniciach: z každej strany, každej hrany, každého vrcholu.

Každá uzavretá oblasť je konvexný mnohosten, ktorý vieme popísať jeho vrcholmi $v_1, v_2, \ldots, v_m \in \mathbb{R}^d$. Všetky body mnohostenu sú konvexné kombinácie vrcholov, teda $\sum \lambda_i v_i$, kde $\lambda_i \geq 0$ a $\sum \lambda_i = 1$. Nekonečné oblasti vieme popísať množinou vrcholov $v_1, \ldots, v_m \in \mathbb{R}^d$ a „lúčov“ $y_1, \ldots, y_\ell \in \mathbb{R}^d$. Body v tejto oblasti sa dajú zapísať v tvare $\sum \lambda_i v_i + \sum \mu_j y_j$, kde $\lambda_i, \mu_j \geq 0$ a $\sum \lambda_i = 1$.

Eliminácia premenných a kvantifikátorov

Druhý možný pohľad na teóriu lineárnej aritmetiky je výpočtový – cez elimináciu premenných a kvantifikátorov. Keďže riešenie sústav lineárnych rovníc a lineárne programovanie sú špeciálnym prípadom lineárnej aritmetiky, poďme sa najskôr pozrieť, ako sa dá robiť eliminácia pri takýchto problémoch.

Sústavy lineárnych rovníc

$$Ax = b$$

sú špeciálnym prípadom formule, ktorá sa skladá iba z existenčných kvantifikátorov, všetky atómy sú rovnosti pospájané spojkou „a zároveň“ ($\wedge$).

Sústavy rovníc sa dajú riešiť dobre známou Gaussovou eliminációu: Vyberieme si premennú, ktorú chceme eliminovať, vyjadríme ju z niektorej rovnice pomocou ostatných premenných:

$$x_d = (a_1 x_1 + a_2 x_2 + \cdots + a_{d-1} x_{d-1} + b)/a_d$$

a tento výraz dosadíme do všetkých ostatných rovníc. Rovnicu pre x_d môžeme zahodiť a pokračovať so zvyškom.

Trochu menej známe už je lineárne programovanie

$$Ax \leq b,$$

čo je v zásade sústava lineárnych nerovníc[1], čo je opäť špeciálny prípad formule iba s existenčnými kvantifikátormi, kde atómy sú neostré nerovnosti ($\leq$) pospájané spojkou „a zároveň“ ($\wedge$).

Hoci sa lineárne programy dajú riešiť v polynomiálnom čase, my si ukážeme menej efektívnu metódu riešenia pomocou tzv. Fourierovej-Motzkinovej eliminácie premenných: Vyberieme si premennú, ktorú chceme eliminovať a vyjadríme ju z každej nerovnice pomocou ostatných premenných. Dostaneme nerovnice tvaru

$$\begin{aligned} x_d &\geq \ell_i(x_1, x_2, \ldots, x_{d-1}) \\ x_d &\leq u_j(x_1, x_2, \ldots, x_{d-1}) \\ &\quad + \text{zvyšné nerovnice, ktoré neobsahujú } x_d \end{aligned}$$

kde ℓ_i, u_j sú lineárne funkcie (tam, kde bol koeficient pri x_d záporný, sa po predelení zmení $\leq$ na $\geq$). Kedy má táto sústava riešenie? Zjavne vtedy (pozri obrázok 15.2a), keď sú všetky horné hranice väčšie alebo rovné ako všetky dolné hranice. Nerovnice s x_d preto môžeme nahradiť $i \times j$ nerovnicami tvaru

$$\ell_i(x_1, x_2, \ldots, x_{d-1}) \leq u_j(x_1, x_2, \ldots, x_{d-1}).$$

V najhoršom prípade dostaneme až $\Theta(n^2)$ nerovníc, zato máme o jednu premennú menej. Po $d-1$ krokoch nám ostane jediná premenná a triviálne nerovnice, z ktorých $[\max_i \ell_i, \min_j u_j]$ definuje interval prípustných hodnôt – program má riešenie práve vtedy, keď je neprázdny. Algoritmus je v najhoršom prípade až exponenciálny, na druhej strane je to inšpirácia, ako by sa mohli rozhodovať všeobecné formuly v lineárnej aritmetike.

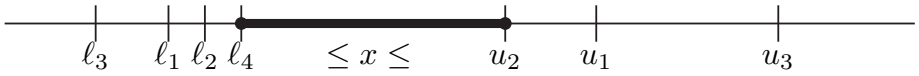
Hlavná myšlienka eliminácie premenných a kvantifikátorov je, že pre každú formulu tvaru $(Q_1 x_1) \cdots (Q_d x_d)\phi(x_1, \ldots, x_d)$ vieme vytvoriť (omnoho dlhšiu, ale) ekvivalentnú formulu *bez* kvantifikátorov ϕ', ktorú triviálne vyhodnotíme. Kvantifikátorov sa budeme zbavovať postupne, *„zvnútra smerom von“*, to znamená v poradí od najvnútornejšieho $(Q_d x_d)$ až po vonkajší kvantifikátor $(Q_1 x_1)$.

Ak je Q_d univerzálny kvantifikátor, môžeme ho zameniť za existenčný, keďže $\forall x : \phi \iff \neg\exists x : \neg\phi$. Predstavme si, že hodnoty $x_1, \ldots, x_{d-1}$ sme už zafixovali. Vyjadrime z každej (ne)rovnice x_d pomocou ostatných premenných:

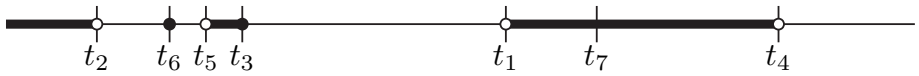
$$x_d \lesseqgtr t_i(x_1, \ldots, x_{d-1}) = c_{i,0} + \sum_{j=1}^{d-1} c_{i,j} x_j$$

Aké hodnoty x_d prichádzajú do úvahy? Situácia je znázornená na obrázku 15.2b. Hodnoty t_i nám rozdelia číselnú os na konečne veľa úsekov, pričom v jednotlivých intervaloch sa pravdivostná hodnota ϕ nemení. Stačí teda za x_d zvoliť „$-\infty$“, „$+\infty$“ (v úvodzovkách, to znamená nejakú hodnotu menšiu/väčšiu ako všetky ostatné), body $t_1, t_2, \ldots, t_7$, alebo „niečo medzi nimi“. Keďže nevieme, v akom poradí sú body t_i (porade bude závisieť od $x_1, \ldots, x_{d-1}$), môžeme vyskúšať body v strede medzi ľubovoľnou dvojicou: $(t_i + t_j)/2$.

[1] v praxi väčšinou nehľadáme len *nejaké* riešenie, ale také, čo maximalizuje nejakú lineárnu funkciu premenných



(a) V lineárnom programovaní potrebujeme splniť všetky nerovnosti tvaru $\ell_i(\vec{x}) \leq x \leq u_j(\vec{x})$ naraz, čo zodpovedá formuli $\bigwedge_i x \geq \ell_i(\vec{x}) \wedge \bigwedge_j x \leq u_j(\vec{x})$. LP má riešenie práve vtedy, ak sú všetky horné hranice väčšie ako všetky dolné.



(b) V teórii $\mathrm{Th}(\mathbb{R}, +)$ sú atomické formuly pospájané logickými spojkami do ľubovoľne zložitej formule ϕ, stále však platí, že ak by sme zafixovali hodnoty všetkých ostatných premenných, horné a dolné hranice rozdelia os x na konečne veľa intervalov, na ktorých sa pravdivostná hodnota nemení. Nemusíme teda skúšať všetky hodnoty x – stačí vyskúšať hodnotu úplne vľavo („$-\infty$"), úplne vpravo („$+\infty$"), jednu z hodnôt $t_i(\vec{x})$ a hodnoty medzi nimi.

Obr. 15.2

To znamená, že vo formule $\phi(x_d) = \phi(x_1, \ldots, x_d)$ nahradíme (ne)rovnice s x_d nasledovne:

$$\exists x_d : \phi(x_d) \iff \phi(-\infty) \vee \phi(+\infty) \vee \bigvee_i \phi(t_i) \vee \bigvee_{i \neq j} \phi\left(\frac{t_i + t_j}{2}\right)$$

(kde t_i myslíme ako skrátený zápis pre $t_i(x_1, \ldots, x_{d-1})$). Formule tvaru $\phi(\pm\infty)$ môžeme ihneď zjednodušiť, keďže

$$\begin{aligned} +\infty \geq \text{hocičo} &\quad\rightsquigarrow\quad \textsc{True}, \\ +\infty \leq \text{hocičo} &\quad\rightsquigarrow\quad \textsc{False}, \\ \pm\infty = \text{hocičo} &\quad\rightsquigarrow\quad \textsc{False}. \end{aligned}$$

Ešte lepšie riešenie dostaneme s pomocou infinitesimálnych (nekonečne malých) čísel (Loos a Weispfenning, 1993). Nech $\varepsilon > 0$ je hodnota menšia ako ľubovoľné kladné reálne číslo, takže ak $t_i < t_j$, tak aj $t_i + \varepsilon < t_j$. Potom

$$\exists x_d : \phi(x_d) \iff \phi(-\infty) \vee \phi(+\infty) \vee \bigvee_i \phi(t_i) \vee \bigvee_i \phi(t_i + \varepsilon)$$

a počet (ne)rovníc bude rásť len lineárne, nie kvadraticky. Formulu vieme následne zjednodušiť a infinitesimálnych hodnôt sa zbaviť, napríklad:

$$\begin{aligned} t_i + \varepsilon = t_j &\quad\rightsquigarrow\quad \textsc{False}, \\ t_i + \varepsilon < t_j &\quad\rightsquigarrow\quad t_i < t_j, \\ t_i + \varepsilon \leq t_j &\quad\rightsquigarrow\quad t_i < t_j, \\ t_i + \varepsilon > t_j &\quad\rightsquigarrow\quad t_i \geq t_j. \end{aligned}$$

V najhoršom prípade nám počet atómov narastie až na dvojnásobok a kým eliminujeme všetky premenné, môže ich byť exponenciálne veľa.

Koľko bitov potrebujeme na zápis riešenia?

Ukázali sme si dva možné prístupy k riešeniu: „geometrický“, kde počítame všetky oblasti a elimináciu kvantifikátorov. Pri oboch prístupoch nám potenciálne môžu vznikať pomerne veľké čísla, respektíve koeficienty sú racionálne čísla s pomerne veľkým čitateľom a menovateľom. Vyvstáva prirodzená otázka, koľko bitov vlastne potrebujeme na ich reprezentáciu? Táto otázka taktiež vedie k tretiemu možnému prístupu:

Pre racionálne číslo $x = \pm p/q$ (kde $p, q \in \mathbb{N}$) definujme veľkosť $||x|| = \max(p, q)$. Číslo x vieme zapísať pomocou $O(\log ||x||)$ bitov. Budeme písať $x \preceq M$, ak $||x|| \leq M$, teda $x = p/q$, kde čitateľ aj menovateľ sú menší ako M. Ak vieme, že všetky racionálne čísla, ktoré vzniknú pri riešení predchádzajúcimi prístupmi, majú čitateľ aj menovateľ maximálne M, potom

$$(Q_1 x_1)(Q_2 x_2) \cdots (Q_d x_d)\phi(x_1, \ldots, x_d)$$
$$\Longleftrightarrow$$
$$(Q_1 x_1 \preceq M)(Q_2 x_2 \preceq M) \cdots (Q_d x_d \preceq M)\phi(x_1, \ldots, x_d)$$

To znamená, že namiesto *neobmedzených* kvantifikátorov $\forall$, $\exists$ stačí vyskúšať konečne veľa racionálnych čísel veľkosti najviac M.

Ukážeme si, že čísla, ktoré nám pri výpočtoch vzniknú, môžu byť až dvojito exponenciálne – to znamená, že len na reprezentáciu jedného takého čísla potrebujeme exponenciálnu pamäť. Z tohto už potom vyplýva, že

Veta 15.2 (Ferrante a Rackoff (1975), Berman (1980)). *Lineárna aritmetika je rozhodnuteľná v* EXPSPACE *a presnejšie v exponenciálnom čase s lineárnym počtom alternácií:*

$$\mathrm{Th}(\mathbb{R}, +, \leq, 0, 1) \in \mathsf{STA}(*, 2^{O(n)}, n).$$

Presburgerova aritmetika

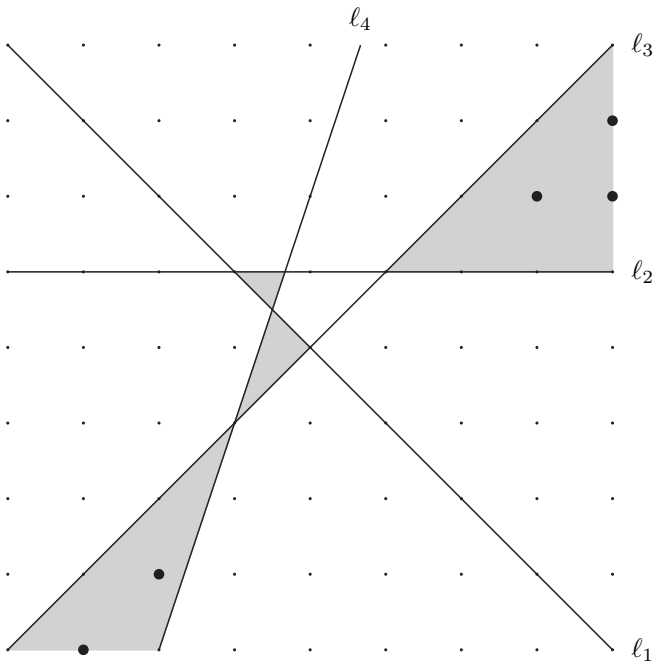
Cooper (1972), analýza Oppen (1978):

1. Zbavíme sa negácie, prepíšeme atómy tak, aby nám ostali iba $<, \setminus, \not\setminus$. (Napríklad $x = a \mapsto a - 1 < x \wedge x < a + 1$.)

2. Nech d je najmenší spoločný násobok všetkých koeficientov pri x. Všetky nerovnice rozšírime, aby bol koeficient x vždy d. Formulu $\exists x : \phi(d \times x)$ teraz môžeme nahradiť $\exists x : d \setminus x \wedge \phi(x)$.

3. Ostávajú nám atómy tvaru a) $x < u_i$, b) $\ell_i < x$, c) $d_i \setminus x + y_i$ a d) $d_i \not\setminus x + y_i$. Nech D je najmenší spoločný násobok všetkých d_i.

$$\exists x : \phi(x) \Longleftrightarrow \bigvee_{k=1}^{D} \phi_{-\infty}(k) \vee \bigvee_{k=1}^{D} \bigvee_{\ell_i} \phi(\ell_i + k)$$



Obr. 15.3

$\phi_{-\infty}(k)$ je formula, kde atómy typu a) nahradíme TRUE, atómy typu b) nahradíme FALSE a vo zvyšku x nahradíme k. Prípadne, ak je atómov typu a) menej ako atómov typu b), výhodnejšie je:

$$\exists x : \phi(x) \Longleftrightarrow \bigvee_{k=1}^{D} \phi_{\infty}(-k) \vee \bigvee_{k=1}^{D} \bigvee_{u_i} \phi(u_i - k)$$

Nelineárna reálna aritmetika

$\mathrm{Th}(\mathbb{R}, +, \cdot, \leq)$ je rozhodnuteľná v EXPSPACE (Ben-Or a spol., 1984).

15.2 Dolné odhady

Reálne čísla

Spravme si malú súťaž: Skúste formulou dĺžky n v lineárnej teórii reálnych čísel, teda pomocou premenných a symbolov $0, 1, +, =, \exists, \forall$, zadefinovať čo najväčšie prirodzené číslo M.

Jedna triviálna možnosť je:

$$\exists M : M = \underbrace{1 + 1 + 1 + \cdots + 1}_{n}.$$

Takto však formulou dĺžky $O(n)$ dosiahneme iba číslo n. Dá sa definovať väčšie číslo?

Samozrejme:

$$\exists M, k_1, \ldots, k_n : k_1 = 1 + 1 \wedge k_2 = k_1 + k_1 \wedge k_3 = k_2 + k_2 \wedge \cdots \wedge M = k_n.$$

Takto formulou dĺžky $O(n)$ dosiahneme číslo veľkosti rádovo $M = O(2^n)$. To je trochu lepšie, ale stále nie dosť dobré. Prídete na to, ako sa dá definovať ešte väčšie číslo?

Riešenie: Nebudeme definovať priamo číslo K, ale pôjdeme na to cez funkcie – definujeme akciu „násobenie číslom K“. Ak už máme funkciu $f(z) = K \times z$ a aplikujeme ju dvakrát, dostaneme novú funkciu $f'(z) = f(f(z)) = K \times (K \times z) = K^2 \times z$, ktorá násobí K^2. Opakovaným umocňovaním na druhú získame dvojito-exponenciálnu funkciu, keďže $\left(2^{2^k}\right)^2 = 2^{2 \times 2^k} = 2^{2^{k+1}}$.

Takýmto spôsobom indukciou definujeme funkcie $f_k : z \mapsto 2^{2^k} \cdot z$. Platí:

$$\begin{aligned} x = f_0(z) &\iff x = z + z \\ x = f_{k+1}(z) &\iff \exists t : x = f_k(t) \wedge t = f_k(z). \end{aligned}$$

Nakoniec M definujeme ako $M = f_k(1)$.

Tento prístup má len jedinú chybičku: v indukčnom kroku v definícii f_{k+1} použijeme f_k *dvakrát* a teda dĺžka formule sa aspoň zdvojnásobí. Na formulu dĺžky $O(n)$ tak môžeme ísť len $\log n$ krokov a vyrobíme opäť len číslo rádovo exponenciálne.

Čo s tým? (Nápoveda: Všimnite si, že sme dosiaľ používali iba existenčné kvantifikátory.)

Spomeňme si na rovnaký trik, aký sme použili pri dôkaze, že QBF je PSPACE-úplný problém. V programovaní sa to volá „code reuse“: kód pre „dačo = f_k(dačo)“ máme zbytočne dvakrát (tzv. „copy pasta“). Pritom to, čo sme chceli, je iba *použiť* f_k dvakrát, ale s rôznymi parametrami (nie definovať dvakrát). Dá sa to takto:

$$x = f_{k+1}(z) \iff \exists t : \forall X, Z : \begin{bmatrix} (X = x \wedge Z = t) \vee \\ (X = t \wedge Z = z) \end{bmatrix} \rightarrow X = f_k(Z).$$

(Čítaj: $X = f_k(Z)$ pre dvojicu $X, Z = x, t$ aj pre dvojicu $X, Z = t, z$.) Takto je kód f_{k+1} len *o konštantu* dlhší ako kód f_k.

Ak konštrukciu n-krát ziterujeme, dostaneme formulu dĺžky $O(n)$, ktorá kóduje $M = O(2^{2^n})$, teda dvojito-exponenciálne číslo – číslo, ktoré má exponenciálne veľa cifier.

Napríklad formula

$$\begin{aligned} \exists M : (\exists t_3 : \forall x_3, z_3 : [(x_3 = M \wedge z_3 = t_3) \vee (x_3 = t_3 \wedge z_3 = 1)] \rightarrow \\ (\exists t_2 : \forall x_2, z_2 : [(x_2 = x_3 \wedge z_2 = t_2) \vee (x_2 = t_2 \wedge z_2 = z_3)] \rightarrow \\ (\exists t_1 : \forall x_1, z_1 : [(x_1 = x_2 \wedge z_1 = t_1) \vee (x_1 = t_1 \wedge z_1 = z_2)] \rightarrow \\ (\exists t_0 : \forall x_0, z_0 : [(x_0 = x_1 \wedge z_0 = t_0) \vee (x_0 = t_0 \wedge z_0 = z_1)] \rightarrow \\ x_0 = z_0 + z_0)))) \end{aligned}$$

definuje $f_4(1) = 2^{2^4} = 65\,536$. (Ak ju čítame odzadu, tak posledný riadok je $x_0 = f_0(z_0)$, predposledný $x_1 = f_1(z_1) = f_0(\underbrace{f_0(z_0)}_{t_0})$, atď., až $M = f_3(\underbrace{f_3(1)}_{t_3})$.)

OK, po tejto rozcvičke prejdime na skutočnú súťaž: Skúste zadefinovať formulu pre násobenie čo najväčších celých čísel.

V $\mathbb{R}_+$ sa, samozrejme, nedá definovať násobenie ľubovoľne veľkých prirodzených čísiel – z toho by totiž vyplývala nerozhodnuteľnosť a $\mathbb{R}_+$ je rozhodnuteľná. Na druhej strane, násobenie celých čísel v nejakom rozsahu, povedzme $[M] = \{0, 1, 2, \ldots, M-1\}$ sa dá definovať – triviálny spôsob je zadrátovať výsledok pre všetky dvojice:

$$x = k \times z, \text{pre } k, z \in [M] \quad \equiv \bigvee_{a,b\in[M], c=a\cdot b} (x = c \wedge k = a \wedge z = b)$$

Motivácia? Podľa dôkazu Gödelovej vety z predchádzajúcej kapitoly vieme s pomocou násobenia popísať výpočty Turingovho stroja. Vieme zadefinovať formulu $\phi_M(w)$, ktorá je pravdivá, ak M akceptuje slovo w. Ak dokážeme formulou dĺžky n definovať násobenie pre $t(n)$-bitové čísla, síce nedokážeme nerozhodnuteľnosť, ale budeme vedieť popísať výpočty TS dĺžky $\Omega(t(n))$. Budeme vedieť zadefinovať formulu $\phi_M(w)$, ktorá je pravdivá, ak M akceptuje slovo w na $c{\cdot}t(n)$ krokov – a ako vieme, problém $\exists w : \phi_M(w)$ je $\mathsf{NTIME}(t(n))$-ťažký.

To znamená, že ak vieme „naprogramovať“ násobenie čísel polynomiálnej dĺžky, problém je NP-ťažký. Ak vieme naprogramovať násobenie čísel exponenciálnej dĺžky, problém je NEXP-ťažký. A ak vieme naprogramovať násobenie čísel dvojito-exponenciálnej dĺžky, problém je $\mathsf{2NEXP}$-ťažký.

Ba čo viac, vieme zostrojiť formulu $\exists w_1 \forall w_2 \cdots Q w_k : \phi_M(w_1 \# w_2 \# \cdots \# w_k)$ a tento problém je $\mathsf{STA}(*, t(n), k)$-ťažký, teda rozhodnuteľnosť bude ťažká pre triedu s lineárne veľa alternáciami.

Takže ako na násobenie? Skúsme použiť rekurziu (s tým, že báza bude niečo ako $x = 0 \times z \equiv x = 0$, $x = 1 \times z \equiv x = z$ a $x = 2 \times z \equiv x = z + z$).

Jednoduchá možnosť je využiť, že

$$(k+1) \times z = k \times z + z,$$

ale takto by sme sa ďaleko nedostali. Lepší spôsob je rozdeliť si úlohu na dva prípady – násobenie párnymi a nepárnymi číslami:

$$(2k) \times z = k \times z + k \times z$$
$$(2k+1) \times z = k \times z + k \times z + 1.$$

Takýmto spôsobom ak vieme násobiť n-bitové čísla (symbol $\times_{(n)}$), indukciou definujeme násobenie $(n+1)$-bitových:

$$x = K \times_{(n+1)} z \quad \equiv \quad \exists k, t : t = k \times_{(n)} z \wedge \begin{bmatrix} (K = k + k \wedge x = t + t) \vee \\ (K = k + k + 1 \wedge x = t + t + z) \end{bmatrix}.$$

Tým sme dokázali NP-ťažkosť rozhodnovania $\mathbb{R}_+$.

Ako dosiahneme násobenie ešte väčších čísel? Majme $2m$-bitové číslo k a označme k_1 prvých m bitov a k_2 druhých m bitov. Inými slovami, napíšme k v tvare $k = 2^m \cdot k_1 + k_2$. Potom

$$k \times_{(2m)} z = (2^m \cdot k_1 + k_2) \times_{(2m)} z = 2^m \cdot (k_1 \times_{(m)} z) + (k_2 \times_{(m)} z).$$

Respektíve pre $m = 2^n$:

$$k \times_{(2^{n+1})} z = (2^{2^n} \cdot k_1 + k_2) \times_{(2^{n+1})} z = 2^{2^n} \cdot (k_1 \times_{(2^n)} z) + (k_2 \times_{(2^n)} z).$$

Pripomeňme, že násobenie číslom 2^{2^n} sú funkcie f_n, ktoré sme definovali vyššie.[2] Takto postupne definujeme násobenie 1-, 2-, 4-, 8-, 16-, 32-, ... bitovým číslom (x ani z nie sú obmedzené – môžu to byť dokonca reálne čísla, ale indukciou vieme dokázať, že k musí byť celé číslo).

Samozrejme, aby formula nebola príliš dlhá, namiesto copy pasty použijeme trik s (∀vstup, výstup) a kód pre $\times_{(2^n)}$ použijeme len raz:

$$\begin{aligned} x = k \times_{(2^{n+1})} z \quad \equiv \quad & \exists k_1, k_2, t_1, t_2 : k = f_n(k_1) + k_2 \wedge x = f_n(t_1) + t_2 \wedge \\ & \forall k, t : \begin{bmatrix} (k = k_1 \wedge t = t_1) \vee \\ (k = k_2 \wedge t = t_2) \end{bmatrix} \rightarrow t = k \times_{(2^n)} z. \end{aligned}$$

Napríklad násobenie 16-bitovým číslom (medzi 0 a 65535) by sme mohli definovať formulou

$$\begin{aligned} & x = w \times_{(16)} z \equiv \\ & \exists b_1, b_2, t_1, t_2 : w = f_3(b_1) + b_2 \wedge x = f_3(t_1) + t_2 \wedge \\ & \quad \forall (b, t) \in \{(b_1, t_1), (b_2, t_2)\} : \exists n_1, n_2, t_3, t_4 : b = f_2(n_1) + n_2 \wedge t = f_2(t_3) + t_4 \wedge \\ & \quad\quad \forall (n, t) \in \{(n_1, t_3), (n_2, t_4)\} : \exists c_1, c_2, t_5, t_6 : n = f_1(c_1) + c_2 \wedge t = f_1(t_5) + t_6 \wedge \\ & \quad\quad\quad \forall (c, t) \in \{(c_1, t_5), (c_2, t_6)\} : (c = 0 \wedge t = 0) \vee (c = 1 \wedge t = z) \vee \\ & \quad\quad\quad\quad\quad\quad (c = 2 \wedge t = z + z) \vee (c = 3 \wedge t = z + z + z). \end{aligned}$$

(Čítaj: slovo, word w sa skladá z dvoch bytov b_1, b_2 a každý byte tvoria dva „nibble“ (4 bity) a každý nibble tvoria dva „crumby“ (2 bity).)

Veta 15.3 (Fischer a Rabin (1998), Berman (1980)). *Teória reálnych čísel so sčítaním* $\mathrm{Th}(\mathbb{R}, +, \leq, 0, 1)$ *je* NEXP*-ťažká, presnejšie* $\mathsf{STA}(*, 2^{O(n)}, n)$*-úplná.*

■ **Dôkaz.** Nech $N = 2^n$ a $I_N = \{0, 1, 2, \ldots, 2^N - 1\}$. Ukázali sme, ako definovať násobenie N-bitovým číslom. Predikát $x \in I_N$, teda x je N-bitové *prirodzené* číslo, defujeme ako $x = x \times_{(N)} 1$. Z toho už dokážeme definovať deliteľnosť ($d \setminus x$, kde $d, x \in I_N$), prvočísla ($\mathbb{P} \cap I_N$), mocniny prvočísel a pracovať s i-tou cifrou N-bitového čísla, či výpočtami dĺžky N bitov. □

[2] Druhá možnosť, bez použitia f_n, je zapísať číslo k v tvare $k_1 \times_{(m)} k_2 + k_2 + k_3$, kde k_1, k_2, k_3 sú m-bitové čísla a

$$\begin{aligned} k \times_{(2m)} z &= (k_1 \times_{(m)} k_2 + k_2 + k_3) \times_{(2m)} z \\ &= k_1 \times_{(m)} (k_2 \times_{(m)} z) + (k_2 \times_{(m)} z) + (k_3 \times_{(m)} z). \end{aligned}$$

Presburgerova aritmetika

Začnime opäť súťažou: Skúste napísať formulu dĺžky n, ktorá definuje čo najväčšie prirodzené číslo M^*.

V predošlej sekcii sme si ukázali formulu pre 2^{2^n} a zdá sa, že väčšie číslo sa ani nedá. Zdanie však klame.

Pripomeňme, že okrem čísla 2^{2^n} sme si zadefinovali násobenie $N = 2^n$-bitových čísiel ... respektíve násobenie N-bitovým číslom(!), lebo vo výraze „$x = k \times z$“ sú x a z neobmedzené – a tento rozdiel je podstatný.

V reálnych číslach, keď sme chceli hovoriť o celých číslach a deliteľnosti, museli sme pridať podmienku $x \in I_N$. Napríklad definícia $d \setminus x$ bola, že $\exists q : x = d \times q$, pričom $q \in I_N$. Podmienka „$q \in I_N$“ bola podstatná, pretože v opačnom prípade by bola pravda aj $7 \setminus 13$, keďže $13 = 7 \times 1.\overline{857142}$ a to sa nám nehodí napríklad ak chceme definovať prvočísla.

Avšak v Presburgerovej aritmetike sú naším univerzom prirodzené čísla – každá premenná je *automaticky* prirodzené číslo a deliteľnosť $d \setminus x$ môžeme definovať ako $\exists q : x = d \times q$, bodka. Praktický dôsledok? Vieme hovoriť o deliteľnosti ľubovoľne veľkých čísel N-bitovým číslom.

A ako nám to pomôže? Označme $\mathbb{P}_N = \mathbb{P} \cap I_N$ množinu N-bitových prvočísel. Číslo M^* bude ich súčin:

$$M^* = \prod_{p \in \mathbb{P}_N} p.$$

Môžeme ho definovať ako *najmenšie* číslo, ktoré je deliteľné všetkými N-bitovými prvočíslami:

$$\exists M^* : \underbrace{(\forall p \in \mathbb{P}_N : p \setminus M^*)}_{\text{deliteľné prvočíslami z } \mathbb{P}_N} \wedge \underbrace{[\forall M' : (\forall p \in \mathbb{P}_N : p \setminus M')}_{\substack{\text{a každé iné číslo} \\ \text{s touto vlastnosťou...}}} \to \underbrace{M* \leq M']}_{\text{... je väčšie}}.$$

Číslo M^* sa prezýva tiež „primoriál“ (ako „faktoriál“, ale násobíme iba prvočísla), a značí sa $M^* = 2^N\# = \prod_{p \leq 2^N} p$. Platí, že $x\# = 2^{\Theta(x)}$, presnejšie[3] $x\# = e^{(1+o(1))x}$, takže pre dostatočne veľké n máme

$$M^* = 2^N\# \geq 2^{2^N} = 2^{2^{2^n}}.$$

Záver: V PA vieme zadefinovať exponenciálne väčšie číslo ako v $\mathbb{R}_+$.

A ako zadefinujeme formulu pre násobenie čo najväčších celých čísel? Použijeme Čínsku zvyškovú vetu. Podľa nej zobrazenie

$$x \bmod M^* \mapsto (x \bmod p_1, \ldots, x \bmod p_k)$$

tvorí izomorfizmus okruhov $\mathbb{Z}_{M^*} \cong \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_k}$. Takže

$$x \equiv y + z \pmod{M^*} \iff x \equiv y + z \pmod{p_1} \wedge \cdots \wedge x \equiv y + z \pmod{p_k} \text{ a}$$
$$x \equiv y \cdot z \pmod{M^*} \iff x \equiv y \cdot z \pmod{p_1} \wedge \cdots \wedge x \equiv y \cdot z \pmod{p_k}.$$

[3] Viď Dusart (2010). Po zlogaritmovaní dostaneme $\ln x\# = \ln \prod_{p \leq x} p = \sum_{p \leq x} \ln p$, čo je funkcia známa ako Čebyševova ϑ. V teórii čísel sa používa pri odhade počtu prvočísel menších ako x.

Ďalej budeme používať konvenciu, že malé písmená (x, p, k, t) označujú premenné $\leq M = 2^N$, teda N-bitové čísla a veľké písmená (X, Y, Z, K, Q, T) označujú premenné $\leq M^*$ s „veľkými“ hodnotami. Takže postupne definujeme

$$\begin{aligned} X \bmod p = x &\equiv \exists Q : X = p \times_{(N)} Q + x \wedge x < p \\ X \equiv Y \pmod p &\equiv X \bmod p = Y \bmod p \\ X \equiv K \times Z \pmod p &\equiv X \equiv (K \bmod p) \times_{(N)} Z \pmod p \\ &\equiv \exists x, k, t, T : X \bmod p = x \wedge K \bmod p = k \wedge \\ &\qquad T = k \times_{(N)} Z \wedge T \bmod p = t \wedge x = t \\ X = K \times Z &\equiv X, Y, Z \leq M^* \wedge \forall p \in \mathbb{P}_N : X \equiv K \times Z \pmod p \end{aligned}$$

Veta 15.4 (Fischer a Rabin (1998), Berman (1980)). *Presburgerova aritmetika* $\mathrm{Th}(\mathbb{N}, +, =, 0, 1)$ *je* $\mathsf{2NEXP}$*-ťažká, presnejšie* $\mathsf{STA}(*, 2^{2^{O(n)}}, n)$*-úplná.*

Jeden algoritmus rozhodujúci Presburgerovu aritmetiku si ukážeme v nasledujúcej kapitole. Ten bude dosť pomalý, existujú však efektívne algoritmy, ktoré sa dajú implementovať v $\mathsf{STA}(*, 2^{2^{O(n)}}, n)$ a dosahujú dolný odhad, ktorý sme si dokázali (pozri Cooper (1972), Oppen (1978)).

Dostávame tak veľmi presnú charakterizáciu problému rozhodnuteľnosti z pohľadu zložitosti. Je dokonca známe, ako počet striedaní kvantifikátorov a počet premenných ovplyvňuje zložitosť: Ak označíme $\mathsf{PrA}(i, j)$ tzv. Σ_i-fragment Presburgerovej aritmetiky, tzn. formuly, ktoré obsahujú i blokov rovnakých kvantifikátorov, prvý blok je existenčný, a každý blok má najviac j premenných, tak problém rozhodnuteľnosti pre

- $\mathsf{PrA}(1, j) \in \mathsf{P}$
- $\mathsf{PrA}(1, *)$, existenčné formuly, je NP-úplný
- $\mathsf{PrA}(i + 1, j)$ je Σ_i^{P}-úplný
- $\mathsf{PrA}(i + 1, *)$ je Σ_i^{EXP}-úplný (kde triedy $\Sigma_i^{\mathsf{EXP}} = \mathsf{STA}(*, 2^{\mathsf{poly}(n)}, i)$ tvoria tzv. exponenciálnu hierarchiu – analogickú k polynomiálnej hierarchii).

Viac: pozri Haase (2018).

Úlohy

- Uvažujme jazyk predikátovej logiky s rovnosťou a operáciami + a ×. Pre danú formulu ϕ a $n \geq 2$, aká je zložitosť problému rozhodnuteľnosti pre okruh $\mathbb{Z}_n$? T.j., aký ťažký je problém rozhodnúť, či $\mathbb{Z}_n \models \phi$?
- (Sontag, 1985) Aký ťažký je problém rozhodnuteľnosti pre lineárnu aritmetiku s *fixným* počtom kvantifikátorov?
- Akú zložitosť má teória reálnych čísel s funkciou sínus $\mathrm{Th}(\mathbb{R}, +, \times, =, \sin)$?

Literatúra

Ben-Or, Michael, Dexter Kozen, a John Reif. 1984. "The complexity of elementary algebra and geometry." In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*. s. 457–464.

Berman, Leonard. 1980. "The complexity of logical theories." *Theoretical Computer Science* 11(1), s. 71–77.

Cooper, David C. 1972. "Theorem proving in arithmetic without multiplication." *Machine intelligence* 7(91-99), str. 300.

Dusart, Pierre. 2010. "Estimates of some functions over primes without RH." *arXiv preprint arXiv:1002.0442* .

Ferrante, Jeanne a Charles Rackoff. 1975. "A decision procedure for the first order theory of real addition with order." *SIAM Journal on Computing* 4(1), s. 69–76.

Fischer, Michael J a Michael O Rabin. 1998. "Super-exponential complexity of Presburger arithmetic." In *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Springer, s. 122–135.

Haase, Christoph. 2018. "A survival guide to Presburger arithmetic." *ACM SIGLOG News* 5(3), s. 67–82.

Loos, Rüdiger a Volker Weispfenning. 1993. "Applying linear quantifier elimination." *The computer journal* 36(5), s. 450–462.

Oppen, Derek C. 1978. "A $2^{2^{2^{p(n)}}}$ upper bound on the complexity of Presburger arithmetic." *Journal of Computer and System Sciences* 16(3), s. 323–332.

Sontag, Eduardo D. 1985. "Real addition and the polynomial hierarchy." *Inf. Process. Lett.* 20(3), s. 115–120.

Kapitola 16

S1S

S1S je druhorádová teória prirodzených čísel s nasledovníkom, $\mathrm{Th}(\mathbb{N}, s, \in)$.

To druhé „S“ je od slova *successor*, nasledovník, čo je funkcia $s : \mathbb{N} \to \mathbb{N}$ daná predpisom

$$n \mapsto n + 1.$$

No a to prvé „S“ je od slova *second order*. V predikátovej logike druhého rádu máme nielen „obyčajné“ premenné $x_1, x_2, x_3, \ldots$, ktoré reprezentujú prirodzené čísla, ale aj množinové premenné $X_1, X_2, X_3, \ldots$. Kvantifikovať môžeme nielen cez čísla, ale aj cez množiny, teda $(\forall X)$ a $(\exists Y)$ znamená „pre každú množinu X (podmnožinu $\mathbb{N}$)“ a „existuje Y, podmnožina $\mathbb{N}$“. Nakoniec jednotka medzi nimi znamená, že pracujeme s unárnou abecedou, čo je ekvivalentné prirodzeným číslam.[1]

Na jednej strane s číslami nevieme robiť žiadne zaujímavé operácie ako sčítanie, či násobenie. K dispozícii máme len veľmi jednoduchú operáciu „zvýšenie o 1“. To túto teóriu značne zjednodušuje. Na druhej strane možnosť kvantifikovať cez celé množiny čísiel dodáva teórii obrovskú silu. My si ukážeme, že teória s nasledovníkom je v skutočnosti rozhodnuteľná, hoci veľmi veľmi ťažká.

Doposiaľ sme definovali len úplne minimalistickú verziu S1S, poďme si najskôr ukázať, že rôzne užitočné relácie vieme dodefinovať:

- $A \subseteq B$ podľa definície, ak $(\forall x)(x \in A \to x \in B)$,
- ... potom $A = B$, ak $A \subseteq B \wedge B \subseteq A$.
- Nulu vieme definovať ako číslo, ktoré nemá predchodcu: $\exists 0 : \neg \exists z : sz = 0$
- a vďaka nej vieme zapísať ľubovoľnú konštantu, napríklad 3 je $s(s(s(0)))$.
- Podobne, výrazy ako $s(s(x))$ budeme skracovať ako $x + 2$.

[1] V S2S pracujeme so slovami nad binárnou abecedou $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ a máme dve následnícke funkcie $\Sigma^* \to \Sigma^*$:

$$s_0(w) = w\mathtt{0} \qquad s_1(w) = w\mathtt{1}.$$

S2S je $\mathrm{Th}(\Sigma^*, s_0, s_1, \in)$.

- Ako definujeme, že dve čísla sa rovnajú? Priamo to nejde, musíme ísť cez množiny. Jedna možnosť je povedať, že $x = y$, ak neexistuje množina, ktorá by obsahovala x a neobsahovala y: $\neg(\exists S : x \in S \wedge y \notin S)$ alebo $\forall S : x \in S \leftrightarrow y \in S$.

  Druhá možnosť je vytvoriť jednoprvkové množiny $\{x\}$ a $\{y\}$, ktoré vieme porovnať. Množinu $\{x\}$ vieme definovať ako „najmenšiu množinu, ktorá obsahuje x“, t.j. $\exists X : x \in X \wedge \forall X' : x \in X' \rightarrow X \subseteq X'$.

  Takúto konštrukciu cez „najmenšiu množinu, ktorá...“ budeme používať pomerne často, takže môžeme zaviesť skratku

  $$X = \min \Phi(x) \equiv \big[\forall x : \Phi(x) \rightarrow x \in X\big] \wedge \big[\forall X' : (\forall x : \Phi(x) \rightarrow x \in X') \rightarrow X \subseteq X'\big].$$

- Vieme napríklad porovnávať, či je $x < y$? Áno, ak symbol $<$ nemáme priamo v jazyku, môžeme ho dodefinovať. Pôjde to trochu okľukou cez množiny, ale je to jednoduchý príklad, na ktorom si ukážeme zopár fínt a silu množín. Definujeme množinu V_x čísel väčších ako x; byť väčší ako x je to isté ako patriť do V_x. Ako? Nuž $s(x) \in V_x$, to je jasné a potom by sme chceli povedať, že aj všetky nasledujúce čísla patria do V_x; inými slovami, V_x je uzavretá na nasledovníka, alebo: $\forall n : n \in V_x \rightarrow s(n) \in V_x$. Má to však jeden háčik: toto spĺňa aj množina všetkých čísel $\mathbb{N}$ – potrebujeme ešte povedať, že „žiadne iné čísla už do V_x nepatria“. Ako na to? Povieme, že „X je najmenšia taká množina“. Inými slovami, ľubovoľná iná množina, ktorá obsahuje $s(x)$ a všetkých nasledovníkov je nadmnožinou V_x:

  $$\Big(\underbrace{s(x) \in V_x}_{\text{obsahuje } x+1} \wedge \underbrace{\forall n : n \in V_x \rightarrow s(n) \in V_x}_{\text{a je uzavretá na nasledovníka}} \Big)$$

  $$\wedge \Big(\forall V' : \underbrace{\big(s(x) \in V' \wedge \forall n : n \in V' \rightarrow s(n) \in V'\big)}_{\text{a každá množina, ktorá toto spĺňa}} \rightarrow \underbrace{V_x \subseteq V'}_{\text{je jej nadmnožinou}} \Big)$$

  Jednoduché porovnanie $x < y$ potom vieme zapísať:

  $$\exists V_x : [\text{tak, ako sme si ju definovali}] \wedge y \in V_x.$$

  Iná, trochu priamejšia možnosť: $x < y$ práve vtedy, keď *každá* množina, ktorá obsahuje $s(x)$ a všetkých následníkov $s(x)$, obsahuje aj y.

  $$\forall X : \Big(s(x) \in X \wedge \big(\forall n : n \in X \rightarrow s(n) \in X\big)\Big) \rightarrow y \in X$$

  Všeobecne by sme mohli definovať konštrukciu uzáveru $\mathrm{Cl}(S, \Phi)$ množiny S na vlastnosť Φ, takto:

  $$C = \mathrm{Cl}(S, \Phi) \equiv C = \min(S \subseteq C \wedge \forall x : x \in C \wedge \Phi(x, y) \rightarrow y \in C).$$

  Množina $V_x = \mathrm{Cl}(\{s(x)\}, y = s(x))$.

- Vďaka tomu napríklad vieme povedať, či je množina konečná:

$$\exists h : \forall x : x \in X \to x < h.$$

Ďalej teda budeme používať symboly $=, \neq, <, \leq, \subseteq$, budeme písať priamo konštanty, dokonca výrazy ako $x + 3$ namiesto $s(s(s(x)))$ s tým, že skratky v týchto formulách sa dajú rozpísať podľa definícií vyššie.

Na množiny sa môžeme pozerať viacerými spôsobmi. Jeden spôsob je ako na predikáty, napríklad ak P je množina párnych čísiel, tak $n \in P$ je predikát „byť párnym číslom“; podobne vyššie sme definovali množinu V_x ako predikát „byť väčší ako x“. Druhý spôsob je ako na nekonečné postupnosti núl a jednotiek, respektíve nekonečné binárne reťazce; na množinu $X \subseteq \mathbb{N}$ sa môžeme pozerať ako na nekonečné slovo $x = x_0x_1x_2\cdots$, kde $x_i = \mathtt{1}$, ak $i \in X$, inak $x_i = \mathtt{0}$.

16.1 S1S je ťažšia ako Presburgerova aritmetika

V S1S vieme zapísať Presburgerovu aritmetiku: premenné budeme reprezentovať množinovými premennými, pričom množina reprezentuje prirodzené číslo, ak jej charakteristická funkcia zapísaná ako reťazec je binárny zápis tohto čísla.

- X reprezentuje 0, ak $\forall z : z \notin X$
- X reprezentuje 1, ak $\forall z : z \in X \leftrightarrow z = 0$
- X reprezentuje číslo $26 = 11010_2$, ak $\forall z : z \in X \leftrightarrow (z = 1 \vee z = 3 \vee z = 4)$
- $X = A + B$: existuje množina C – prenos do vyššieho rádu

0	1	0	1	1	0	1	– A
0	1	1	0	1	1	0	– B
1	1	1	1	0	0		– C, prenos do vyššieho rádu
1	1	0	0	0	1	1	– X

$$\text{„}\exists C : C_0 = 0 \wedge \forall i : C_{i+1} \leftrightarrow A_i + B_i + C_i > 1 \wedge X_i = A_i \oplus B_i \oplus C_i\text{“}$$

Z toho vyplýva, že problém rozhodnuteľnosti S1S je aspoň 2NEXP-ťažký.

Skúste si rozmyslieť či/ako by sa dalo implementovať násobenie, teda nájsť formulu pre $X = A \cdot B$.

16.2 S1S nie je elementárna

V skutočnosti je to s S1S oveľa oveľa horšie: S1S sa nedá rozhodovať nielen v exponenciálnom čase (2^n), ani dvojito-exponenciálnom čase (2^{2^n}), ba ani trojito-exponenciálnom čase ($2^{2^{2^n}}$),... S1S sa nedá dokonca rozhodovať v čase

$$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}}_{k}$$

pre ľubovoľné fixné konečné k.

Inými slovami, ak definujeme triedu elementárne rekurzívnych problémov

$$\begin{aligned}\mathsf{ELEMENTARY} &= \mathsf{EXP} \cup \mathsf{2EXP} \cup \mathsf{3EXP} \cup \cdots \\ &= \mathsf{DTIME}(2^n) \cup \mathsf{DTIME}(2^{2^n}) \cup \mathsf{DTIME}(2^{2^{2^n}}) \cup \cdots\end{aligned}$$

tak tvrdíme, že S1S $\notin$ ELEMENTARY.

Napriek tomu, ako si ukážeme neskôr v tejto kapitole, S1S rozhodnuteľná je!

Pre kontext: funkcia, ktorá rastie rýchlejšie ako všetky elementárne rastie naozaj veľmi veľmi rýchlo, ale stále je to nič napríklad v porovnaní s Ackermannovou funkciou, či Busy Beaverom. Rýchlo rastúce funkcie sa dobre zapisujú pomocou tzv. Knuthovej šípkovej notácie. Základom je iterácia (opakovanie):

- násobenie je len opakované sčítanie:

$$2 \times n = \underbrace{2 + (2 + (\cdots + 2))}_{n}$$

- podobne umocnenie je len opakované násobenie:

$$2^n = 2 \uparrow n = \underbrace{2 \times (2 \times (\cdots \times 2))}_{n}$$

- definujme teraz analogicky vežu výšky n, alias „dvojšípku“, ako opakované umocnenie:

$$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2^2}}}}}}_{n} = 2 \uparrow\uparrow n = \underbrace{2 \uparrow (2 \uparrow (\cdots \uparrow 2))}_{n}$$

- a takto môžeme pokračovať ďalej: a „trojšípka“ n je len n-krát iterovaná „dvojšípka“ a-čok:

$$2 \uparrow\uparrow\uparrow n = \underbrace{2 \uparrow\uparrow (2 \uparrow\uparrow (\cdots \uparrow\uparrow 2))}_{n}$$

  $2 \uparrow\uparrow\uparrow n$ je veža dvojok výšky $2 \uparrow\uparrow\uparrow (n-1)$.

- ... vo všeobecnosti „k-šípku“ definujeme pomocou $(k-1)$-šípky:

$$2 \uparrow^k n = \underbrace{2 \uparrow^{k-1} (2 \uparrow^{k-1} (\cdots \uparrow^{k-1} 2))}_{n}$$

Predstavme si programy s jednoduchými for-cyklami (bez while-cyklov či rekurzie). Pre konkrétnosť uvažujme programy s premennými $x_0, x_1, x_2, \ldots$ (vstup aj výstup budú v x_0), pričom jediné povolené inštrukcie sú $x_i \leftarrow 0$, $x_i \leftarrow x_j$, $x_i \leftarrow x_i + 1$ a „opakuj x_i-krát { program }“. Všimnite si, že takýto program vždy skončí. Dá sa dokázať, že ak nepovolíme vnorené cykly, najväčšia hodnota,

ktorú dokážu vypočítať je $\leq c \cdot n$. Ak povolíme 2 vnorené cykly, najväčšia hodnota bude $\leq c^n$ a pri troch vnorených cykloch to bude $\leq c \uparrow\uparrow n$. Vo všeobecnosti program s k vnorenými cyklami vypočíta najviac hodnotu $\leq c \uparrow^{k-1} n$, kde c je konštanta.

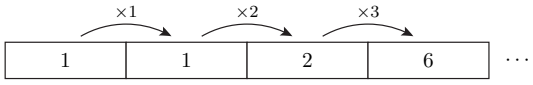
My si ukážeme, ako formulou dĺžky $O(n)$ v S1S popísať číslo $2 \uparrow\uparrow n$ a aritmetiku takto veľkých čísiel, z čoho vyplýva, že dokážeme popísať výpočet TS a otázku, či v takomto čase stroj zastaví. Všetky elementárne funkcie sú vežičky exponentov fixnej výšky, zatiaľčo $2 \uparrow\uparrow n$ je veža exponentov, ktorá rastie s n, je teda od určitého n väčšia.

Pre kontext: Primitívne rekurzívne funkcie sú v zásade tie, ktoré sa dajú vypočítať programami s jednoduchými for-cyklami (bez while-cyklov či rekurzie). Diagonála Ackermannovej funkcie rastie zhruba ako $2 \uparrow^n n$ a teda je väčšia (od určitého n) ako akákoľvek primitívne rekurzívna funkcia a nedá sa vypočítať žiadnym for-cyklovým programom. Napriek tomu sa dá jednoducho spočítať, ak povolíme while-cykly (alebo všeobecnú rekurziu). Busy Beaver $B(n)$ je funkcia definovaná ako najväčší počet jednotiek, ktoré vie vypísať n-stavový TS (nad binárnou abecedou) na prázdnom vstupe. Táto funkcia rastie tak rýchlo, že je nevypočítateľná. (Ak by existoval TS, ktorý ju počíta, vedeli by sme ho jednoducho upraviť, aby spočítal $B(n)+1$, čiže hodnotu väčšiu, ako vypočíta ľubovoľný n-stavový TS; pre $n =$ počet jeho stavov by však musel vypočítať hodnotu väčšiu ako sám vypočíta a to sa nedá.)

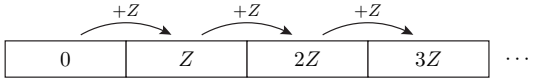
Veta 16.1 (Meyer (1975)). *Rozhodnuteľnosť S1S nie je elementárne rekurzívna.*

■ **Dôkaz.** Vráťme sa k našej súťaži „aké najväčšie číslo vieme zadefinovať na n znakov“. Ako v predchádzajúcej kapitole budeme postupovať iteratívne, avšak kým v $\mathbb{R}_+$ sme pridaním pár znakov vedeli zadefinovať číslo zhruba dvakrát dlhšie, tu pridaním pár znakov vytvoríme exponeniálne väčšie číslo.

Základom bude definovať reláciu $i \equiv j \pmod n$ pre čoraz väčšie n. Ukážeme si, že ak toto máme, budeme vedieť pracovať s n-bitovými blokmi. Budeme vedieť pracovať s množinovou premennou ako s postupnosťou blokov, kde každý blok je jedno n-bitové číslo. A ak vieme kódovať postupnosti, vieme definovať zaujímavé funkcie – spomeňme si, ako Gödel definoval faktoriál cez „existuje postupnosť taká, že každý ďalší prvok…“:

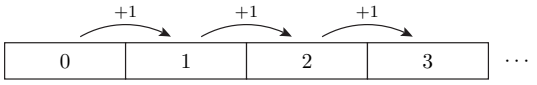


Rovnakým spôsobom budeme definovať násobenie n-bitových čísel:



Súčin $K \times Z$ spočítame pomocou opakovaného pričítania. Na konci sa stačí pozrieť na K-ty blok. Ako na to?

Vyrobíme si „n-bitové počítadlo“ – postupnosť $0, 1, 2, 3, \ldots$



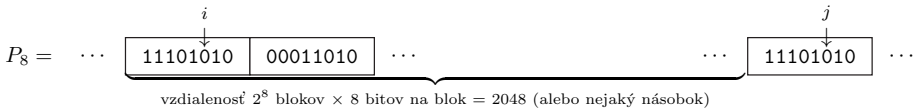
potom „K-ty blok“ znamená „ten istý blok, čo má na počítadle hodnotu K“.

Pripomeňme, že všetky operácie budú n-bitové, teda modulo 2^n a v skutočnosti zadefinujeme nekonečnú periodickú postupnosť, napríklad n-bitové počítadlo P_n bude periodická postupnosť

$$0, 1, 2, 3, \ldots, 2^n - 1, \quad 0, 1, 2, 3, \ldots, 2^n - 1, \quad 0, 1, 2, 3, \ldots$$

Takže tvrdíme, že ak vieme definovať $i \equiv j \pmod{n}$, budeme vedieť definovať prácu s n-bitovými blokmi a následne aritmetiku n-bitových čísel. Ostáva ukázať, ako sa posunúť na vyššiu úroveň a ako definovať $i \equiv j \pmod{N}$ pre $N \gg n$.

Tu nám pomôže práve n-bitové počítadlo. Rovnaké bloky sa totiž opakujú vždy po 2^n blokoch, teda ak máme dva rovnaké bloky, ich vzdialenosť (v blokoch) musí byť deliteľná 2^n. Predstavme si i a j ako indexy do P_n.



Ak sú dva bloky, kam i a j ukazujú, rovnaké a navyše i aj j ukazujú na rovnaký bit v rámci bloku, potom ich vzdialenosť musí byť deliteľná $n \cdot 2^n$. Napríklad na obrázku máme 8-bitové počítadlo; i aj j ukazujú na rovnaký blok „`11101010`“, takže počet *blokov* medzi nimi je násobok $2^8 = 256$. Navyše i aj j ukazujú na piaty bit v rámci bloku, preto ich vzdialenosť *v bitoch* je deliteľná $8 \cdot 256 = 2048$. Inými slovami, $i \equiv j \pmod{2048}$. Vo všeobecnosti takto z $i \equiv j \pmod{n}$ vyrobíme $i \equiv j \pmod{n \cdot 2^n}$.

Poďme postupne:

- Reláciu $i \equiv j \pmod{1}$ nemusíme ani definovať, je triviálne pravdivá pre každé i a j. Odtiaľ sa odpichneme.
- Jednobitové počítadlo P_1 je postupnosť `0101010101` $\cdots$ definujeme ju predpisom $0 \notin P_1 \wedge \forall i : i \in P_1 \leftrightarrow s(i) \notin P_1$.
- Na základe toho vieme definovať $i \equiv j \pmod{2}$ ako $i \in P_1 \leftrightarrow j \in P_1$
- Dvojbitové počítadlo `00 10 01 11 00 10` $\cdots$ by sme ešte mohli definovať priamo: $0 \notin P_1 \wedge 1 \notin P_1 \wedge \forall i : i \equiv 0 \pmod{2} \rightarrow [(i + 2 \in P_2 \leftrightarrow i \notin P_2) \wedge (i + 3 \in P_2 \leftrightarrow i \in P_2 \oplus i + 1 \in P_2)]$, a odtiaľ $i \equiv j \pmod{8}$ ak $i \equiv j \pmod{2}$ a nech i_0, j_0 je začiatok bloku a $(i_0 \in P_2 \leftrightarrow j_0 \in P_2) \wedge (i_0 + 1 \in P_2 \leftrightarrow j_0 + 1 \in P_2)$.

Ďalej už však poďme všeobecne. Predpokladajme, že vieme definovať $i \equiv j \pmod{n}$ a P_n a ukážme, ako definovať $i \equiv j \pmod{N}$ a P_N pre $N = n \cdot 2^n$.

- Definujme funkciu $\mathrm{zb}(i)$ – začiatok bloku i; je to $i - (i \bmod n)$ alebo najväčšie $z \leq i$ deliteľné n, respektíve také, že medzi z a i už nie je nič deliteľné n:

  $$z = \mathrm{zb}(i) \iff z \leq i \wedge z \equiv 0 \pmod{n} \wedge \forall j : z < j < i : j \not\equiv 0 \pmod{n}.$$

  Vďaka tomu vieme povedať napríklad, že dva indexy i, j ukazujú na ten istý blok – je to vtedy, keď $\mathrm{zb}(i) = \mathrm{zb}(j)$. A vďaka tomu vieme povedať napríklad „pre všetky indexy z jedného bloku platí ...“

- Množiny A, B budeme teraz vnímať ako postupnosti n-bitových blokov. Definujme „$A[x] = B[y]$“ (tvrdenie „blok, kam ukazuje x v A, je rovnaký ako blok, kam ukazuje y v B“):

  $$\forall i, j : i \equiv j \pmod{n} \wedge \mathrm{zb}(i) = \mathrm{zb}(x) \wedge \mathrm{zb}(j) = \mathrm{zb}(y) \to (i \in A \leftrightarrow j \in B).$$

- Z toho už vieme definovať $x \equiv y \pmod{N}$:

  $$x \equiv y \pmod{n} \wedge P_n[x] = P_n[y].$$

- Mimochodom samotné číslo N vieme definovať ako najmenšie číslo deliteľné N väčšie ako nula. Pričítanie N vieme definovať ako najmenšie väčšie číslo s rovnakým zvyškom mod N.

- „A je N-bitové číslo“ práve vtedy, keď všetky bity od N-tého vyššie sú nulové:

  $$\forall i : i \geq N \to i \notin A$$

  alebo

  $$\forall i : i \in A \to \mathrm{zb}(i) = 0.$$

- V predchádzajúcej sekcii sme si definovali sčítanie. Sčítanie N-bitových čísel, teda modulo 2^N definujeme jednoducho tak, že výsledok orežeme:

  $$X = A + B \bmod 2^N \iff \exists X' = A + B \wedge \underbrace{\forall i : i < N : i \in X \leftrightarrow i \in X'}_{\text{prvých } N \text{ bitov sa zhoduje}} \wedge \underbrace{\forall i : i \geq N : i \notin X}_{\text{zvyšné bity sú nulové – } X \text{ je } N\text{-bitové číslo}}$$

- Nakoniec P_N definujeme ako postupnosť, ktorá začína nulou a každé ďalšie číslo je o 1 väčšie (modulo 2^N):

  $$(\forall i : i < N \to i \notin P_N) \wedge \forall i : P_N[i + N] = P_N[i] + 1 \bmod 2^N.$$

Takto vieme po k iteráciach definovať $M > \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{k}$-bitové čísla. Násobenie $X = K \times Z \bmod 2^M$ definujeme cez postupnosť násobkov:

$$\exists T : T[0] = 0 \wedge \forall i : T[i + M] = T[i] + Z \bmod 2^M \wedge T[K] = X,$$

kde $T[K] = X$ je skratka pre

$$\exists j : P_M[j] = K \wedge j \text{ je najmenšie také} \wedge T[j] = X.$$

V S1S tak vieme sformulovať aritmetiku M-bitových čísel a to, že „TS akceptuje/zastaví na $\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n}$ krokov“ a preto rozhodnuteľnosť S1S $\notin$ ELEMENTARY.

□

Stockmeyer a Meyer (2002) dokázali zostrojiť konkrétnu formulu Ψ v S1S na rozhodnuteľnosť $\leq$ 610-znakových formúl EWS1S treba obvod s $> 10^{125}$ hradlami na rozhodnuteľnosť $\leq$ 614-znakových formúl s úspešnosťou $\geq 2/3$ treba obvod s $> 10^{125}$ hradlami 610 znakov = 3660 bitov

Pre porovnanie, počet atómov v pozorovateľnom vesmíre sa odhaduje niekde medzi 10^{78} a 10^{82}.

16.3 Dyadická S1S je nerozhodnuteľná

Doposiaľ sme uvažovali takzvanú *monadickú* logiku S1S, kde môžeme kvantifikovať cez množiny (napríklad $\exists X \subseteq \mathbb{N}$). V *dyadickej* S1S môžeme kvantifikovať aj cez binárne relácie, teda $\exists R \subseteq \mathbb{N}^2$. Pochopiteľne tým pádom dokážeme kvantifikovať aj cez funkcie $\exists F : \mathbb{N} \to \mathbb{N}$ – funkcie sú totiž iba relácie, pre ktoré platí $\forall x : \exists y : (x, y) \in F \wedge \forall y' : (x, y') \in F \to y' = y$.

Veta 16.2. *Teória dyadickej S1S je nerozhodnuteľná.*

■ **Dôkaz.** Spomeňme si na Postov korešpondenčný problém (PKP): Máme niekoľko typov domín, napríklad

typ 1: $\begin{array}{|c|}\hline \texttt{a} \\ \hline \texttt{baa} \\ \hline \end{array}$ typ 2: $\begin{array}{|c|}\hline \texttt{ab} \\ \hline \texttt{aa} \\ \hline \end{array}$ typ 3: $\begin{array}{|c|}\hline \texttt{bba} \\ \hline \texttt{bb} \\ \hline \end{array}$,

z každého typu máme ľubovoľný počet kópií. Úlohou je zistiť, či sa dá vytvoriť konečná postupnosť domín tak, že keď ich postavíme vedľa seba, horný reťazec bude rovnaký ako spodný. V našom malom príklade sa to dá, jedno riešenie je:

bba	ab	bba	a
bb	aa	bb	baa

Nasledujú po sebe dominá typu 3, 2, 3, 1; hore aj dolu je reťazec `bbaabbbaa`. Vo všeobecnosti sú dané dvojice $(t_1, b_1), \ldots, (t_d, b_d)$ a riešenie je postupnosť $i_1, i_2, \ldots, i_k$ taká, že

$$s = t_{i_1} t_{i_2} \cdots t_{i_k} = b_{i_1} b_{i_2} \cdots b_{i_k}.$$

Tento problém je nerozhodnuteľný (dá sa dokázať, že pre ľubovoľný TS vieme vyrobiť takú sadu domín, že riešenie kóduje jeho výpočet a teda existuje práve vtedy, keď TS zastane). Ukážeme, že PKP sa dá redukovať na dyadickú S1S.

Predveďme si to na našom príklade. Riešenie vyššie si môžeme zapísať aj takto:

b b a	a b	b b a	a
b b	a a	b b	b a a

Zodpovedajúce písmená sme posunuli pod seba a tým sa nám posunuli aj hranice medzi dominami. Napríklad druhé domino začína hore na pozícii 3, ale dolu na pozícii 2. Znázorniť to môžeme aj takto:

	↓			↓		↓			↓	↓
pozície	0	1	2	3	4	5	6	7	8	9
$s =$	b	b	a	a	b	b	b	a	a	
	↑		↑		↑		↑			↑

kde s je výsledný reťazec a šípky ukazujú, kde začínajú jednotlivé dominá hore a dolu.

$S = \{0, 1, 4, 5, 6\}$

- S: spoločný text hore aj dolu (presnejšie pozície béčok)
- H_i, D_i, T_i: pozícia, kde začína i-te domino hore/dolu; typ i-teho domina

$$\begin{aligned}
&\exists \text{množina } S : \exists \text{funkcie } H, D, T : H_0 = 0 \wedge D_0 = 0 \\
&\quad \wedge \forall i : \big((T_i = 1 \to [\psi_{\mathtt{a}}(H_i) \wedge \psi_{\mathtt{baa}}(D_i) \wedge H_{i+1} = H_i + 1 \wedge D_{i+1} = D_i + 3]) \wedge \\
&\qquad\quad (T_i = 2 \to [\psi_{\mathtt{ab}}(H_i) \wedge \psi_{\mathtt{aa}}(D_i) \wedge H_{i+1} = H_i + 2 \wedge D_{i+1} = D_i + 2]) \wedge \\
&\qquad\quad (T_i = 3 \to [\psi_{\mathtt{bba}}(H_i) \wedge \psi_{\mathtt{bb}}(D_i) \wedge H_{i+1} = H_i + 3 \wedge D_{i+1} = D_i + 2])\big) \\
&\quad \wedge \exists i : i \neq 0 \wedge H_i = D_i
\end{aligned}$$

□

16.4 S1S a konečné automaty na nekonečných slovách

Vráťme sa však k monadickej S1S. Každá formula $\phi(X)$ s jednou voľnou množinovou premennou ako parametrom, prirodzene definuje množinu tých X, pre ktoré je

formula pravdivá. A ako sme si povedali, na množiny sa môžeme pozerať ako na nekonečné binárne reťazce. Formula $\phi(X)$ teda definuje „jazyk“ nekonečných slov $L_\phi = \{\chi_A \mid \mathbb{N} \models \phi(A)\}$.

V tejto kapitole vybudujeme teóriu formálnych jazykov a automatov na nekonečných slovách. Ukážeme si, ako rozšíriť definíciu konečných automatov či regulárnych výrazov a ukážeme, že existuje analógia regulárnych jazykov aj pre nekonečné slová. Túto triedu jazykov voláme ω-regulárne (ω-REG) a dokážeme si, že sú to presne tie jazyky, ktoré sa dajú popísať S1S formulami.

Omega-regulárne jazyky

Nech Σ je konečná abeceda; Σ^ω budeme značiť množinu *nekonečných slov* nad abecedou Σ. Podmnožiny Σ^ω voláme jazyky, presnejšie ω-jazyky.[2]

Definícia 16.1. *Nech $L \subseteq \Sigma^+$ je jazyk (konečných slov). Definujme L^ω (analogicky ku L^*) ako jazyk, kde každé slovo je zreťazenie nekonečne veľa slov z L, teda $L^\omega = \{x_0 x_1 \ldots \mid x_i \in L - \{\varepsilon\}\}$.*

Zreťazenie dvoch nekonečných slov nemá zmysel, ale môžeme zreťaziť konečné slovo s nekonečným: ak $w = w_1 \cdots w_n$ a $x = x_1 x_2 x_3 \cdots$, tak zreťazenie $wx = w_1 \cdots w_n x_1 x_2 x_3 \cdots$ a pre $L \subseteq \Sigma^*, N \subseteq \Sigma^\omega$, je $L \cdot N = \{wx \mid w \in L, x \in N\}$.

Definícia 16.2 (Omega-regulárne výrazy a jazyky). *Definujeme induktívne. Ak α je (klasický) regexp, tak*

- *α^ω je ω-regulárny výraz a $L(\alpha^\omega) = L(\alpha)^\omega$,*

a ak γ, δ sú ω-regulárne, tak aj

- *$\alpha\gamma$ je ω-regulárny a $L(\alpha\gamma) = L(\alpha) \cdot L(\gamma)$,*
- *$(\gamma \mid \delta)$ je ω-regulárny výraz a $L(\alpha \mid \beta) = L(\alpha) \cup L(\beta)$.*

Jazyky, ktoré popisujú ω-regulárne výrazy tvoria triedu ω-regulárnych jazykov.

Použitím distributívneho zákona ľahko vidíme, že každý ω-regulárny jazyk je konečným zjednotením ω-jazykov tvaru AB^ω, kde $A, B \subseteq \Sigma^*$ sú regulárne.

Napríklad $(\mathtt{0} \mid \mathtt{1})^*\mathtt{0}^\omega$ je jazyk slov, ktoré obsahujú len konečne veľa jednotiek (od istého miesta idú samé nuly). Naopak $(\mathtt{0}^*\mathtt{1})^\omega$ je jazyk slov, ktoré obsahujú nekonečne veľa jednotiek.

Konečné automaty

Klasické regulárne výrazy sú ekvivalentné konečným automatom, je preto prirodzené pokúsiť sa nájsť podobný model aj pre ω-regulárne jazyky. Nech $A = (Q, \Sigma, \delta, q_0, F)$ je klasický nedeterministický konečný automat. Môžeme ho spustiť aj na nekonečnom slove $x = x_0 x_1 x_2 \cdots$ a dostaneme tak nekonečný výpočet – postupnosť stavov $\rho = q_0 q_1 q_2 \ldots$ takú, že $q_{i+1} \in \delta(q_i, x_i)$. Jediný problém

[2] Symbolom ω sa označuje najmenšie nekonečné ordinálne číslo – odtiaľ názvy ako ω-jazyky, ω-regulárne, atď.

je, ako zvoliť nejaké rozumné akceptačné kritérium. Pre konečné slová to bolo jednoduché – automat dočítal slovo a skončil v nejakom stave; ak bol stav akceptačný, slovo akceptoval. Lenže nekonečné výpočty nekončia, žiadny stav nie je posledný, ako teda definujeme akceptáciu?

Existuje viacero možností, ale asi najjednoduchšia je povedať, že výpočet je akceptačný, ak prejde nekonečne veľakrát nejakým akceptačným stavom. Takýto model voláme (nedeterministický) *Büchiho automat*. Formálnejšie:

Definícia 16.3 (Büchiho automat (NBA)). *Nech $A = (Q, \Sigma, \delta, q_0, F)$ je nedeterministický konečný automat. Definujme* $\mathrm{io}(\rho)$ *ako množinu stavov, ktoré výpočet ρ navštívi nekonečne veľakrát; teda* $\mathrm{io}(\rho) = \bigcap_{n\geq 0}\{q_i \mid i \geq n\}$.

Büchiho podmienka: Výpočet ρ je akceptačný, ak $\mathrm{io}(\rho) \cap F \neq \emptyset$. *Jazyk nekonečných slov, pre ktoré existuje akceptačný výpočet, značíme $L_\omega(A)$ a A voláme Büchiho automat.*

Ak neuvedieme inak, pod „Büchiho automatom“ máme vždy na mysli *nedeterministický* Büchiho automat. Determinické Büchiho automaty sú slabšie, definujú menšiu triedu ω-jazykov a nemajú niektoré uzáverové vlastnosti.

Veta 16.3 (Büchi (1962)). *Jazyky akceptované nedeterministickými Büchiho automatmi sú práve ω-regulárne jazyky.*

■ **Náznak dôkazu.** Jedným smerom stačí dokázať, že ak L je regulárny a L_1, L_2 sú akceptované Büchiho automatmi, tak existujú Büchiho automaty pre L^ω, $L \cdot L_1$ a $L_1 \cup L_2$.

Druhým smerom označme $L_{p,q}$ jazyk (konečných) slov, na ktoré sa automat vie dostať zo stavu p do q. Potom tvrdíme, že Büchiho automat akceptuje práve jazyk $\bigcup_{f\in F} L_{q_0,f} L_{f,f}^\omega$. Jazyky $L_{p,q}$ sú zjavne regulárne a teda jazyk akceptovaný Büchiho automatom je ω-regulárny. □

S1S popisuje ω-regulárne jazyky

Veta 16.4 (Büchi (1962)). *Pre každý Büchiho automat M nad abecedou $\{0, 1\}$ existuje v S1S formula $\phi(X)$ a naopak, pre každú S1S formulu $\phi(X)$ existuje Büchiho automat M taký, že $L_\omega(M) = L_\phi$. (Veta sa dá ľahko zovšeobecniť pre viac voľných premenných a väčšie abecedy.)*

■ **Dôkaz.** Ukážeme, ako pre formulu ϕ zostrojíme ekvivalentný NBA. (Opačný smer prenecháme ako úlohu pre čitateľa.)

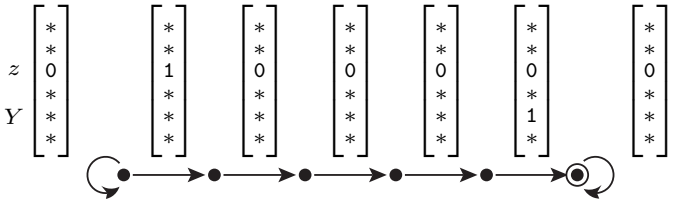
Predstavme si najskôr, že máme formulu ϕ bez kvantifikátorov a všetky číselné aj množinové premenné sú súčasťou vstupu. Pre k premenných bude abeceda $\{0,1\}^k$. Napríklad pre premenné $x = 2$, $y = 0$, $z = 5$, $X = \{0, 2, 4, 6, 8, \ldots\} =$ párne čísla, $Y = \{2, 3, 5, 7, 13, \ldots\} =$ prvočísla a $Z = V_x = \{3, 4, 5, 6, 7, \ldots\}$ by

sme mali vstup

	0	1	2	3	4	5	6	7	8	9	
$x =$	0	0	1	0	0	0	0	0	0	0	
$y =$	1	0	0	0	0	0	0	0	0	0	
$z =$	0	0	0	0	0	1	0	0	0	0	$\cdots$
$X =$	0	0	1	0	1	0	1	0	1	0	
$Y =$	1	0	1	1	0	1	0	1	0	0	
$Z =$	0	0	0	1	1	1	1	1	1	1	

kde každý stĺpec je jeden symbol a každá premenná je jeden riadok. Samozrejme, v riadku, ktorý zodpovedá číselnej premennej, musí byť práve jedna jednotka.

Postupujeme klasicky štrukturálnou indukciou: Atomické formuly sú tvaru $s^k(x) \in X$, pre ktoré ľahko vyrobíme aj deterministický Büchiho automat. Napríklad automat pre $s(s(s(z)))) \in Y$ by vyzeral zhruba takto:



kde $*$ znamená, že na tejto pozícii môže byť ľubovoľná hodnota. Teda čakáme v počiatočnom stave, kým nenarazíme na symbol, kde má z jednotku, odpočítame si pozíciu o 4 ďalej a skontrolujeme, že daný prvok patrí do Y – na danej pozícií je jednotka v riadku pre Y.

Ak už máme NBA A, B pre ϕ, ψ, potom $\phi \vee \psi$, $\phi \wedge \psi$ a $\neg\phi$ popisujú postupne jazyky $L_\omega(A) \cup L_\omega(B)$, $L_\omega(A) \cap L_\omega(B)$ a $\overline{L_\omega(A)} = \Sigma^\omega - L_\omega(A)$. Z toho zjednotenie je ľahké (ako pri NKA), prienik je pomerne ľahký (ale treba si dať pozor, nie je to rovnaká konštrukcia kartézskym súčinom ako pri NKA) a komplement je príšerne ťažký, ale možný – budeme sa mu venovať nižšie. Na tomto mieste sa mu však vieme vyhnúť ak pomocou de Morganovych zákonov posunieme negáciu až k atomickým formulám: zostrojiť automat pre $s^k(x) \notin X$ je jednoduché.

V skutočnosti na vstupe nemáme hodnoty všetkých premenných, ale skoro všetky (až na jednu) sú viazané kvantifikátormi. Pre formuly tvaru $\exists Z : \phi(Z)$ budeme nedeterministicky tipovať množinu/nekonečné slovo Z a zároveň simulovať NBA pre ϕ, tzn. konštrukcia bude ako predtým, avšak „riadok“ pre Z nebudeme čítať zo vstupu, ale budeme ho tipovať pomocou nedeterminizmu. Podobne pre formuly tvaru $\exists z : \phi(z)$ budeme tipovať riadok pre z, pričom zabezpečíme, že obsahuje práve jednu jednotku. Nakoniec formuly tvaru $\forall Z : \phi(Z)$ riešime ako $\neg\exists Z : \neg\phi(Z)$. Tu už sa negácii nevyhneme a veľkosť automatu bude veľmi závisieť od počtu striedaní kvantifikátorov. □

Dôsledok 16.1. *Teória S1S je rozhodnuteľná.*

■ **Dôkaz.** Pre uzavretú S1S formulu ϕ vieme zostrojiť Büchiho automat, ktorý ani nevníma vstup (nedeterministicky tipuje hodnoty premenných) a ak je ϕ pravdivá, vždy akceptuje a ak je nepravdivá, nikdy neakceptuje. Ako zistíme, ktorý prípad nastal? Automat akceptuje práve vtedy, keď existuje tzv. laso – cesta z počiatočného stavu do nejakého akceptačného stavu f a zároveň okružná cesta z f do f. Toto sa dá skontrolovať prehľadávaním grafu stavov v polynomiálnom čase (dokonca nedeterministicky v logaritmickej pamäti). □

Veta 16.5 (Büchi (1962), Sistla a spol. (1987)). *Büchiho automaty, respektíve ω-regulárne jazyky sú uzavreté na komplement. Ak A má n stavov, potom existuje automat pre $\overline{L_\omega(A)}$ s $2^{O(n^2)}$ stavmi.*

■ **Dôkaz.** Nech A je Büchiho automat pre L. Budeme písať $p \overset{u}{\rightsquigarrow} q$, ak na slovo u môže automat prejsť zo stavu p do q a $p \overset{u}{\rightsquigarrow}_F q$, ak navyše cestou prejde cez nejaký akceptačný stav.

Definujeme reláciu ekvivalencie $\approx_A$ na slovách Σ^+; povieme, že slová u, v sú ekvivalentné, ak sa pre každé dva stavy p, q dá dostať z p do q na u práve vtedy, keď sa dá dostať z p do q na v a zároveň z p do q sa dá dostať cez nejaký akceptačný stav na slove u práve vtedy, keď sa to dá na slove v. Teda $u \approx_A v$ ak

$$\forall p, q : (p \overset{u}{\rightsquigarrow} q \iff p \overset{v}{\rightsquigarrow} q) \wedge (p \overset{u}{\rightsquigarrow}_F q \iff p \overset{v}{\rightsquigarrow}_F q).$$

Inými slovami, pre slovo w definujme funkciu $f_w(p) = (Q_1, Q_2)$, ktorá pre každý stav vráti množinu stavov $Q_1 = \Delta(p, w)$, kam sa automat dostane na slovo w a množinu stavov Q_2, kam sa automat dostane cez nejaký akceptačný stav. Potom $v \approx_A w$ práve vtedy, keď $f_v = f_w$. Keďže takýchto funkcií $f : Q \to 2^Q \times 2^Q$ je len konečne veľa, relácia $\approx_A$ má len konečne veľa tried ekvivalencie (konečný index). Navyše každá trieda ekvivalencie $[w]_{\approx_A}$ je regulárny jazyk.

Každé nekonečné slovo w patrí do $R \cdot S^\omega$ pre nejaké triedy ekvivalencie $\approx_A$. Označme $w_{i,j} = w_i w_{i+1} \cdots w_{j-1}$. Predstavme si nekonečný kompletný graf, kde vrcholy sú prirodzené čísla. Pre každú triedu ekvivalencie $\approx_A$ budeme mať jednu farbu a hranu z i do $j > i$ „ofarbíme“ triedou ekvivalencie $[w_{i,j}]_{\approx_A}$. Potom podľa nekonečnej Ramseyho vety tento graf obsahuje nekonečnú jednofarebnú kliku. Nech $i_1, i_2, i_3, \ldots$ sú jej vrcholy. Potom w_{0,i_1} patrí do triedy $R = [w_{0,i_1}]_{\approx_A}$ a všetky $w_{i_k,i_{k+1}}$ patria do tej istej triedy ekvivalencie $S = [w_{i_1,i_2}]_{\approx_A}$. Takže $w \in R \cdot S^\omega$.

Pre každé dve triedy ekvivalencie R, S, je jazyk $R \cdot S^\omega$ buď celý pod L, alebo celý mimo L. Z toho vyplýva, že L aj $\overline{L}$ sa dá napísať ako zjednotenie konečne veľa ω-regulárnych jazykov tvaru $R \cdot S^\omega$. Nech $w \in L \cap R \cdot S^\omega$ a nech ρ je akceptačný výpočet na w. Ukážeme, že každé $w' \approx_A w$ tiež patrí do L. Slovo w sa dá napísať v tvare $w_0 w_1 w_2 \cdots$, kde $w_0 \in R$ a $w_i \in S$ pre $i > 0$. Podobne $w' = w'_0 w'_1 w'_2 \cdots$, kde $w'_0 \in R$ a $w'_i \in S$ pre $i > 0$. Nech q_{i+1} je stav v behu ρ, po načítaní $w_0 \cdots w_i$ a I je množina indexov, pre ktoré $q_i \overset{w_i}{\rightsquigarrow}_F q_{i+1}$ (takýchto je nekonečne veľa). Potom akceptačný výpočet ρ' na slove w' začne v rovnakom stave ako ρ; keďže $w_i \approx_A w'_i$, existuje výpočet z q_i do q_{i+1} a ak $i \in I$, existuje výpočet, ktorý prejde cez F. Takže $w' \in L$. □

Dôsledok 16.2. *Teória S1S je rozhodnuteľná v čase* $2 \uparrow\uparrow O(n)$.

■ **Dôkaz.** Postupujeme podľa konštrukcie z vety 16.4. Každá negácia spôsobí exponenciálny nárast stavov (najviac o 2 vyššiu mocninovú vežu). □

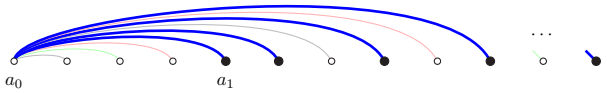
Safra (1988) neskôr ukázal efektívnejšiu konštrukciu automatu s $n^{O(n)} = 2^{O(n \log n)}$ stavmi. Tento výsledok je optimálny až na konštantu v exponente – dá sa ukázať, že pre niektoré automaty na ich komplement potrebujeme $n^{\Omega(n)} = 2^{\Omega(n \log n)}$ stavov. Všimnite si, že to je viac ako v prípade komplementu obyčajných konečných automatov, kde v najhoršom prípade treba a zároveň stačí 2^n stavov.

Ďalší výskum a snaha zmenšiť rozdiel medzi dolným a horným odhadom kulminovali prácami Yan (2006) a Schewe (2009), ktorí dokázali, že v najhoršom prípade treba a zároveň stačí na komplement približne $(0.76n)^n$ stavov. Horný odhad je len $O(n^2)$-krát väčší ako dolný.

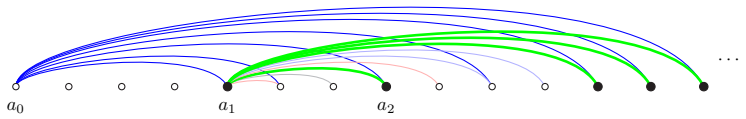
Veta 16.6 (Nekonečná Ramseyho veta). *Predstavme si nekonečný kompletný graf, kde vrcholy sú prirodzené čísla a hrany ofarbime k (konečne veľa) farbami. Nech tento graf ofarbíme akokoľvek, vždy bude obsahovať nekonečnú jednofarebnú kliku.*

■ **Dôkaz.** Predstavme si, že graf je orientovaný „zľava doprava“ a hrana (x, y) pre $x < y$ ide z x do y.

Začnime s $A_0 = \mathbb{N}$, $a_0 = 0$; z vrcholu a_0 vychádza nekonečne veľa hrán, ale máme len konečne veľa farieb. Preto z a_0 vychádza nekonečne veľa hrán niektorej farby (povedzme modrej).



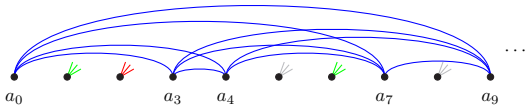
Označme A_1 vrcholy spojené s a_0 modrou hranou a nech $a_1 = \min A_1$. Ostatné vrcholy zahoďme. Z a_1 opäť vychádza nekonečne veľa hrán, takže nejaká farba sa musí opakovať nekonečne veľakrát (povedzme zelená).



Vrcholy z A_1 spojené s a_1 zelenou hranou dáme do A_2 a pokračujeme s $a_2 = \min A_2$. Ostatné vrcholy zahodíme. Stále nám ostáva nekonečný počet vrcholov, takže môžeme pokračovať.

Takto postupne získame nekonečnú postupnosť vrcholov $a_0, a_1, a_2, \ldots$ takú, že všetky hrany vychádzajúce z a_i majú rovnakú farbu f_i:

Keďže farieb je len konečne veľa, niektorá sa v postupnosti $f_0, f_1, f_2, \ldots$ vyskytuje nekonečne veľakrát (povedzme modrá).



Ak vezmeme vrcholy, z ktorých vedú modré hrany, dostávame nekonečnú jednofarebnú kliku. □

Úlohy

- Skúste implementovať násobenie celých čísiel v S1S.
- Dokážte, že monadická druhorádová logika $(\mathbb{N}, <)$ (teda namiesto funkcie successor máme reláciu menší ako) má tú istú silu ako S1S.
- Dokážte, že neexistuje *deterministický* Büchiho automat pre $(\mathtt{0} \mid \mathtt{1})^*\mathtt{1}^\omega$. Sú teda slabšie ako nedeterministické.
- Trieda (tradičných) regulárnych jazykov je uzavretá na prienik. Pre konečné automaty sa dôkaz spravil pomocou kartézskeho súčinu automatov $A_1 \times A_2$: $Q = Q_1 \times Q_2$, $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a)$ a $F = F_1 \times F_2$. Pre Büchiho automaty však táto konštrukcia nefunguje! Nájdite protipríklad – Büchiho automaty A_1, A_2 také, že $L_\omega(A_1 \times A_2) \neq L_\omega(A_1) \cap L_\omega(A_2)$. Platí aspoň jedna inklúzia?
  (ω-regulárne jazyky *sú* uzavreté na prienik, ale automat pre prienik treba skonštruovať inak.)
- Dokážte vetu o limite: Pre $R \subseteq \Sigma*$ nech $\lim R = \{w \in \Sigma^\omega \mid \exists^\infty n \in \mathbb{N} : w_{0\ldots n} \in R\}$. ω-jazyk L je ω-regulárny práve vtedy, keď existuje regulárny R taký, že $L = \lim R$.
- Uvažujme Presburgerovu aritmetiku $\mathrm{Th}(\mathbb{N}, +, V_2)$ s pridaním funkcie $V_2(n) \equiv$ „najvyššia mocnina 2, ktorá delí n“. Je táto teória rozhodnuteľná?

Literatúra

Büchi, J Richard. 1962. “On a decision method in restricted second order arithmetic.” In *Logic, Methodology and Philosophy of Science (Proc. Internat. Congr.)*. Stanford Univ. Press, Stanford, Calif, s. 1–12.

Meyer, Albert R. 1975. “Weak monadic second order theory of succesor is not elementary-recursive.” In *Logic colloquium*. Springer, s. 132–154.

Safra, Shmuel. 1988. “On the complexity of ω-automata.” In *Proc. 29th IEEE Symp. Found. of Comp. Sci.* s. 319–327.

Schewe, Sven. 2009. "Büchi complementation made tight." *arXiv preprint arXiv:0902.2152* .

Sistla, A Prasad, Moshe Y Vardi, a Pierre Wolper. 1987. "The complementation problem for Büchi automata with applications to temporal logic." *Theoretical Computer Science* 49(2-3), s. 217–237.

Stockmeyer, Larry a Albert R Meyer. 2002. "Cosmological lower bound on the circuit complexity of a small problem in logic." *Journal of the ACM (JACM)* 49(6), s. 753–784.

Yan, Qiqi. 2006. "Lower bounds for complementation of ω-automata via the full automata technique." In *International Colloquium on Automata, Languages, and Programming*. Springer, s. 589–600.

Časť VI

Derandomizácia (Treba nám náhodu?)

Úvod

Vieme s pomocou náhody riešiť viac problémov ako deterministicky? Nakoľko nám randomizácia pomáha?

Náhodné čísla majú uplatnenie v rôznych oblastiach informatiky. V kryptológii potrebujeme voliť náhodné kľúče, aby ich útočník nedokázal predpovedať a ak tajnú správu zo-xor-ujeme so skutočne náhodnou postupnosťou, dostaneme perfektne bezpečnú šifru. V teórii hier uvažujeme pravdepodobnostné (zmiešané) stratégie. V dátových štruktúrach vieme vďaka (pseudo-)náhode šetriť pamäť za cenu približných výsledkov (Bloom filtre, streamové algoritmy). Presne spočítať všetky hlasy vo voľbách je pomerne drahá záležitosť, ale odhadnúť výsledok v prieskume verejnej mienky je nepomerne jednoduchšie. Podobne dokážeme približne (ale s ľubovoľnou presnosťou v polynomiálnom čase) spočítať objem mnohorozmerného konvexného telesa.

Nás bude zaujímať otázka časovej zložitosti: Vieme niektoré problémy riešiť rýchlejšie pravdepodobnostne ako deterministicky?

Zdá sa, že odpoveď je áno. Jeden spôsob ako sa pozerať na pravdepodobnostné algoritmy je, že poľavujeme z našich nárokov: netrváme na postupe, ktorý funguje *vždy*, stačí aby fungoval väčšinou (pri väčšine hodov mincou). Je preto prirodzené očakávať, že najlepší pravdepodobnostný algoritmus bude aspoň trochu lepší ako ten, ktorý má náhodu zakázanú. Pre veľmi obmedzené modely to vieme aj dokázať, napríklad hľadanie mediánu sa dá pravdepodobnostne pomocou $1.5n$ porovnaní, ale každý deterministický algoritmus potrebuje viac ako $2n$ porovnaní. To je, samozrejme, veľmi malý rozdiel a otázka znie, či vieme pomocou náhodnosti dosiahnuť aj väčšie, povedzme exponenciálne (alebo aspoň superpolynomiálne) zrýchlenie. Alebo naopak dokážeme všetky algoritmy efektívne „derandomizovať“, tzn. zbaviť sa náhodnosti a vytvoriť ekvivalentný, nie oveľa pomalší deterministický algoritmus?

V súčasnosti odpoveď na túto otázku nepoznáme: teoreticky je stále možné, hoci nepravdepodobné, že $\mathsf{BPP} = \mathsf{EXP}$, teda že všetko, čo vieme vypočítať v exponenciálnom čase, vieme s pomocou náhody vyriešiť v polynomiálnom čase.

Jeden z hlavných problémov, ktoré vieme riešiť rýchlo randomizovane a nedokážeme rýchlo deterministicky je testovanie rovnosti polynómov (nad nejakým poľom). Napríklad, je pravda, že

$$(ux + zvy)^2 + z(vx - uy)^2 \stackrel{?}{=} (u^2 + zv^2)(x^2 + zy^2)$$

alebo

$$1024x^{10}+(x+\sqrt{3})^{10}+(x-\sqrt{3})^{10}+(\sqrt{3}x+1)^{10}+(\sqrt{3}x-1)^{10} \stackrel{?}{=} 1512(x^2+1)^5-1024.$$

Mohli by sme skúsiť zátvorky roznásobiť a jednoducho porovnať koeficienty:

$$u^2x^2 + u^2y^2z + v^2x^2z + v^2y^2z^2$$

$$1512x^{10} + 7560x^8 + 15120x^6 + 15120x^4 + 7560x^2 + 488$$

avšak vo všeobecnosti narazíme na to, že členov môže byť veľmi veľa, zatiaľčo na vstupe môže byť polynóm zapísaný úsporne pomocou aritmetickej formule alebo obvodu[3]. Ak máme k dispozícii skutočne náhodné čísla, stačí dosadiť a vyhodnotiť formulu/obvod na náhodnom vstupe – $p = q$ práve vtedy, keď $p-q = 0$ a nulový polynóm je na každom vstupe nula, zatiaľčo nenulové polynómy na veľa vstupoch dajú nenulový výsledok. Bez toho, aby sme zatiaľ zachádzali do detailov, akonáhle trafíme nenulovú hodnotu, vieme, že polynóm je nenulový, naopak, ak vyskúšame veľa náhodných vstupov a výsledok je vždy nulový, s veľkou pravdepodobnosťou je polynóm nulový (rovnosť platí). Otázka znie, či by sme nevedeli vstupy vygenerovať aj deterministicky (pseudonáhodne) a či by dokázali zaručiť, že neexistuje nenulový polynóm, ktorému by sme ako na potvoru triafali práve jeho korene.

Ďalší (trochu umelý) problém, ktorý nevieme riešiť deterministicky v polynomiálnom čase je Capp – circuit approximation probability problem. Daný je booleovský obvod C a úlohou je aspoň približne spočítať počet vstupov, ktoré obvod akceptuje, alebo ekvivalentne, úlohou je odhadnúť pravdepodobnosť, že obvod akceptuje náhodný vstup. Ak by sme vedeli odhadnúť $\Pr_r[C(r) = 1]$ povedzme s presnosťou ± 0.1, vedeli by sme rozhodovať, či ľubovoľný BPP-algoritmus akceptuje daný vstup – pravdepodobnosť, že taký algoritmus akceptuje vs. neakceptuje, sa totiž líši o viac ako 0.2.

problém	najlepšie známe riešenie	
	pravdepodobnostne	deterministicky
medián	$1.5n$	$> 2n$ porovnaní
minimálna kostra	$O(m)$	$O(m\alpha(n))$
minimálny rez	$\tilde{O}(n^2)$	$O(n^3)$
test prvočíselnosti	$\tilde{O}(n^2)$	$\tilde{O}(n^6)$
lineárne programovanie	$O(d^2n + 2^{O(\sqrt{d \log d})})$	$O(d^{O(d)}n)$
3-SAT	$\tilde{O}(1.308^n)$	$\tilde{O}(1.334^n)$
test rovnosti polynómov	$O(n)$	EXP

Tabuľka 16.1: Porovnanie najlepších známych pravdepodobnostných a deterministických riešení pre rôzne problémy.
??

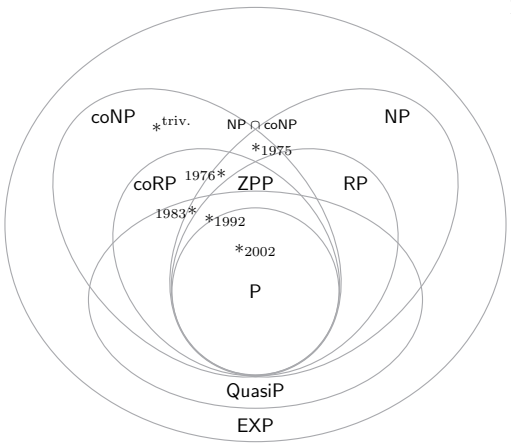
[3]Aritmetické obvody definujeme podobne ako tie booleovské ako acyklické grafy s tým, že hradlá aritmetických obvodov môžu byť konštanty, sčítanie a násobenie.

4

Na druhej strane máme viacero úspešných príkladov derandomizácie. Asi najznámejší je testovanie prvočíselnosti. Tento problém má zaujímavú históriu (pozri obr. 16.1). Zjavne ak je číslo zložené, existuje krátky dôkaz tohto tvrdenia (stačí jeden netriviálny deliteľ); už nie tak zrejmé je, že existujú aj krátke dôkazy prvočíselnosti. Podľa Lucasovej vety, ak pre nejaké a je $a^{n-1} \equiv 1 \pmod{n}$ a pre každé prvočíslo q deliace $n-1$ je $a^{(n-1)/q} \not\equiv 1 \pmod{n}$, tak n je prvočíslo. Certifikát prvočíselnosti potom obsahuje číslo a, prvočíselný rozklad $n-1$ a rekurzívne dôkaz, že každý člen rozkladu je naozaj prvočíslo.

Podľa Fermatovej vety, ak n je prvočíslo, tak $a^{n-1} \equiv 1 \pmod{n}$ (pre $0 < a < p$). Toto sa dá použiť ako jednoduchý test zloženosti: zvolíme náhodné a; ak nespĺňa kongruenciu, n musí byť zložené. Tento test má isté problémy (viď Carmichaelove čísla), ale dajú sa opraviť (viď Rabin-Millerov test) a dá sa dokázať, že ak je n zložené, náhodne zvolené a to odhalí s pravdepodobnosťou aspoň 3/4.

Deterministický AKS test je založený na zovšeobecnenej Fermatovej vete: Nech $n \geq 2$ a čísla a a n sú nesúdeliteľné; potom n je prvočíslo práve vtedy, keď $(x+a)^n \equiv x^n + a \pmod{n}$. (Pre $x = 0$ dostávame Fermatov test.) Mimochodom ďalší možný pravdepodobnostný test prvočíselnosti je zvoliť napríklad $a = 1$ a otestovať rovnosť polynómov $(x+1)^n \equiv x^n + 1 \pmod{n}$ – ak by sme vedeli derandomizovať testovanie rovnosti polynómov, dostaneme nový deterministický algoritmus pre testovanie prvočíselnosti. Agrawal a spol. namiesto toho dokázali, že stačí pre vhodné r vyskúšať len polynomiálne veľa a-čok a rovnosť polynómov testovať modulo $x^r - 1$.



PRIMES ∈

- coNP (triviálne)
- NP (Pratt, 1975)
- P, ak platí rozšírená Riemannova hypotéza (Miller 1976)
- coRP (Rabin 1976)
- coRP (Solovay–Strassen 1977)
- QuasiP (Adleman–Pomerance –Rumely 1983)
- ZPP, ak platí Cramérova hypotéza (Goldwasser–Kilian 1986)
- ZPP (Adleman–Huang 1992)
- P (Agrawal–Kayal–Saxena 2002)
  pôvodný alg.: $\tilde{O}(\beta^{11.5})$
  Lenstra–Pomerance: $\tilde{O}(\beta^6)$
  kde β je dĺžka vstupu (počet bitov)

Obr. 16.1: Pokrok pri riešení problému testovania prvočíselnosti.

[4] Zatiaľčo O-notácia zanedbáva konštanty, $\tilde{O}$ zanedbáva polylogaritmické členy, tzn. $\tilde{O}(f(n)) = O(f(n) \cdot \mathsf{polylog}(f(n)))$, napríklad $\tilde{O}(n^c) = O(n^c \log^k n)$ a $\tilde{O}(2^{cn}) = O(2^{cn} \cdot n^k)$ pre nejaké k.

Druhý veľký úspech bolo derandomizovanie algoritmu pre hľadanie cesty v neorientovanom grafe s malým priestorom (tzv. USTCON). Nájsť cestu medzi dvoma vrcholmi je jednoduché v lineárnom čase štandardným prehľadávaním do šírky či do hĺbky. Tie však potrebujú aj lineárnu pamäť na označovanie navštívených vrcholov. Otázka znie, či sa dá nájsť cesta aj s malým – logaritmickým priestorom. (Len označenie jedného vrcholu zaberá $\log n$ bitov, takže v $O(\log n)$ pamäti si môžeme dovoliť pamätať len konštante veľa vrcholov naraz a nedokážeme si pamätať, ktoré vrcholy sme už navštívili.)

Existuje primitívny randomizovaný algoritmus: začneme v jednom vrchole a hýbeme sa náhodne (v každom vrchole si vyberieme náhodného suseda) až kým nedorazíme do cieľa. Zároveň si počítame počet krokov a ak sa do cieľa nedostaneme ani na $16n^3$ krokov, prehlásime, že cesta neexistuje. Teda USTCON $\in$ RL. Zo Savitchovej vety vieme, že existuje algoritmus v $O(\log^2 n)$ pamäti. Nisan–Szemerédi–Wigderson našli algoritmus s $O(\log^{1.5} n)$ a Armoni a spol. s $O(\log^{4/3} n)$ pamäťou. Úplná derandomizácia sa nakoniec podarila Reingoldovi, ktorý v roku 2004 dokázal, že USTCON $\in$ L. Ekvivalentne, majme d-regulárny graf a v každom vrchole označme vychádzajúce hrany číslami od 1 po d. Pre takto označkovaný graf môžeme každý reťazec nad abecedou $\{1, \ldots, d\}$ chápať ako návod, po ktorej hrane máme ísť v každom vrchole. *Univerzálna postupnosť* je návod, ktorý garantuje, že navštívime všetky vrcholy bez ohľadu na počiatočný vrchol, samotný graf, či jeho označkovanie. Z pravdepodobnostného algoritmu vyplýva, že takéto postupnosti existujú – náhodne zvolená dostatočne dlhá postupnosť má nenulovú pravdepodobnosť, že je univerzálna. Reingoldov výsledok ukazuje, že takéto postupnosti sa dajú zostrojiť v logaritmickom priestore.

Mimochodom, otázka, či $\mathsf{L} \stackrel{?}{=} \mathsf{RL}$ a či sa dajú derandomizovať všetky RL-algoritmy je stále otvorená (vieme však, že $\mathsf{RL} \subseteq \mathsf{DSPACE}(\log^{1.5} n)$). Podobný problém STCON pre *orientované* grafy je NL-úplný a nepoznáme ani randomizovaný algoritmus s logaritmickou pamäťou. Nápad s náhodnou prechádzkou nefunguje, pretože v niektorých grafoch sa na niektoré miesta dostaneme len s exponenciálne malou pravdepodobnosťou, čo je málo.

Do tretice spomeňme úspech s perfektným párovaním, pričom nás zaujíma rýchle paralelné riešenie (NC – polylogaritmický čas s polynomiálne veľa procesormi). Majme graf G. K nemu zostrojme maticu A tak, že ak (i, j) je hrana grafu, tak na políčku $A_{i,j}$ bude premenná $x_{i,j}$ (pre každú hranu máme inú premennú) a na políčku $A_{j,i}$ bude $-x_{i,j}$; ak medzi i, j nie je hrana, tak $A_{i,j} = 0$. Dá sa dokázať, že G má perfektné párovanie práve vtedy, keď $\det A$ je nenulový polynóm.[5] Úlohu sme teda redukovali na testovanie rovnosti polynómov. Stačí teda za $x_{i,j}$ dosadiť náhodné hodnoty (povedzme zo $\mathbb{Z}_p^*$ pre dostatočne veľké prvočíslo) a vyhodnotiť determinant – to dokážeme v RNC. Deterministický NC algoritmus vynašli Eppstein a Vazirani o viac ako 30 rokov neskôr, v roku 2018.

[5] Myšlienka dôkazu v skratke: Determinant je suma cez všetky permutácie π, pričom člen je nenulový, ak všetky (i, π_i) sú hrany v G. Stačí teda počítať len súčet cez všetky cyklové pokrytia. A ak máme cyklové pokrytie s cyklom nepárnej dĺžky, zmenením orientácie dostaneme člen s opačným znamienkom, tzn. všetky takéto členy sa navzájom vyškrtajú a ostanú len cyklové pokrytia, kde každý cyklus má párny počet hrán – a z tohto sa dá ľahko vybrať perfektné párovanie.

Vráťme sa však k časovej zložitosti. Čo vieme povedať o triede BPP? Triviálne $\mathsf{BPP} \subseteq \mathsf{EXP}$, resp. $\mathsf{BPP} \subseteq \mathsf{PSPACE}$. V predchádzajúcich kapitolách sme si ukázali, že BPP je v polynomiálnej hierarchií (Sipser–Gacsova veta: $\mathsf{BPP} \subseteq \Sigma_2^{\mathsf{P}}$) a hody mincou vieme zameniť za polynomiálnu radu (Adlemanova veta: $\mathsf{BPP} \subseteq \mathsf{P/poly}$). Lepšie výsledky momentálne nie sú známe. Napríklad je otvorený problém, či $\mathsf{BPP} \subseteq \mathsf{NP}$, alebo či vieme BPP-problémy rozhodovať čo i len v sub-exponenciálnom čase.

Napriek tomu veríme, že BPP sa dá úplne derandomizovať a všetky BPP-problémy vieme riešiť – síce možno v horšom – ale stále polynomiálnom čase. Inými slovami, veríme, že

$$\mathsf{P} = \mathsf{BPP}.$$

A prečo by sme mali? O tom si povieme v nasledujúcich kapitolách. Ukážeme, že výsledok $\mathsf{P} = \mathsf{BPP}$ vyplýva z čoraz slabších a slabších predpokladov.

Konkrétne, postupne si dokážeme, že:

1. Ak vieme približne odhadovať pravdepodobnosť, že daný obvod akceptuje (riešiť problém CAPP), potom vieme derandomizovať.

2. Ak existuje „dobrý“ pseudonáhodný generátor (PNG), potom vieme odhadnúť pravdepodobnosť, že BPP-algoritmus akceptuje. Intuitívne, dobrý PNG je taký, že ak vymeníme skutočne náhodné bity za pseudonáhodné, nikto si to nevšimne. Existuje však dobrý PNG? Veríme, že áno. Ukážeme, že:

3. Ak existuje „pekelne ťažká“ funkcia, dokážeme z nej vyrobiť dobrý PNG. Intuitívne, pekelne ťažká funkcia sa nedá vypočítať oveľa lepšie ako že náhodne tipujem výsledok. Ak by sme jednoducho tipovali správnu odpoveď, máme polovičnú šancu, že sa trafíme. Respektíve, ak by sme na vstupoch dĺžky n odpovedali buď vždy ÁNO, alebo vždy NIE, tak v jednom z týchto prípadov budeme odpovedať správne pre aspoň polovicu vstupov. Pekelne ťažká funkcia je taká, že ak chceme odpovedať správne na čo i len o chlp viac ako polovici vstupov, potrebujeme exponenciálne veľké obvody. Na prvý pohľad toto znie ako pomerne odvážna hypotéza, my si však ďalej ukážeme, ako sa z ťažkých funkcií dajú skonštruovať ešte ťažšie:

4. Ak existuje „veľmi ťažká“ funkcia (je ťažké ju spočítať na 99% správne), vieme z nej spraviť pekelne ťažkú funkciu (je ťažké ju spočítať na čo i len o chlp viac ako 50%). Existencia veľmi ťažkých funkcií už neznie tak prekvapujúco, pôjdeme však ešte ďalej:

5. Ak existuje ťažká funkcia (v klasickom zmysle, t.j., na jej výpočet potrebujeme exponenciálne veľký obvod), tak z nej dokážeme vyrobiť veľmi ťažkú funkciu (na ktorú potrebujeme exponenciálne veľký obvod, hoci ju stačí spočítať na 99% správne).

Takže ak

$$\exists \text{ ťažká funkcia} \Longrightarrow \exists \text{ veľmi ťažká funkcia} \Longrightarrow \exists \text{ pekelne ťažká funkcia}$$
$$\Longrightarrow \exists \text{ dobrý PNG} \Longrightarrow \text{vieme odhadnúť Pr[BPP-algoritmus akceptuje]}$$
$$\Longrightarrow \mathsf{P} = \mathsf{BPP}.$$

Pod ťažkou funkciou máme pritom na mysli funkciu, ktorá sa nedá spočítať obvodom veľkosti $2^{\gamma n}$ pre nejaké $\gamma > 0$. Náš dôkaz však bude vyžadovať funkciu, ktorá sa dá vypočítať v čase $2^{O(n)}$, teda funkciu, ktorá patrí do $\mathsf{E} = \mathsf{DTIME}(2^{O(n)})$. Môže to byť napríklad funkcia, ktorá sa dá vypočítať v čase 2^{1000n}, ale nedá sa spočítať obvodom veľkosti $2^{0.001n}$.

Pripomeňme, že vďaka jednoduchému počítaciemu argumentu (koľko je všetkých obvodov veľkosti s? a koľko je všetkých funkcií $f : \{0,1\}^n \to \{0,1\}$?) vidíme, že existujú funkcie, ktoré sa nedajú spočítať ani len obvodom veľkosti $2^n/10n$ a skoro všetky funkcie potrebujú obvody väčšie ako $2^{0.001n}$. Dokonca to platí, aj keď nám stačí funkciu spočítať len povedzme na 90%. Ak by sme teda vybrali náhodnú funkciu (pre každý vstup zvolíme náhodnú odpoveď 0 alebo 1), takmer na 100% bude táto funkcia veľmi ťažká. Na derandomizáciu však potrebujeme konkrétnu funkciu z E.

Ak by to ešte stále niekoho nepresvedčilo, tak v skutočnosti aj z menej ako exponenciálne ťažkých funkcií vyplýva aspoň čiastočná derandomizácia. Pripomeňme, že podľa Kannanovej vety už Σ_2^{P} obsahuje funkcie, ktoré nemajú obvod polynomiálnej veľkosti – toto na derandomizáciu nestačí. Z funkcií, ktoré potrebujú superpolynomiálne obvody už však vieme derandomizovať. Konkrétne:

- Ak existuje $f \in \mathsf{E}$, ktorá potrebuje obvod superpolynomiálnej veľkosti ($n^{\alpha(n)}$, kde $\alpha(n)$ môže rásť ľubovoľne pomaly, ale s rastúcim n rastie do nekonečna), tak BPP-problémy vieme riešiť v subexponenciálnom čase, teda lepšie ako v čase 2^{n^ε} pre ľubovoľne malé $\varepsilon > 0$: $\mathsf{BPP} \subseteq \mathsf{SUBEXP} = \bigcap_{\varepsilon>0} \mathsf{DTIME}(2^{n^\varepsilon})$.

- Ak existuje $f \in \mathsf{E}$, ktorá potrebuje obvod veľkosti 2^{n^γ} (pre $\gamma > 0$), napríklad $2^{\sqrt[100]{n}}$, tak $\mathsf{BPP} \subseteq \mathsf{QuasiP} = \mathsf{DTIME}(2^{\mathsf{polylog}(n)})$. Kvázipolynomiálna zložitosť je $n^{\log^k n} = 2^{\log^{k+1} n}$ pre nejaké k. Je to viac ako $n^{O(1)}$, ale menej ako $2^{\Omega(n)}$ (exponent rastie do nekonečna, ale iba polylogaritmicky, nie lineárne).

- Nakoniec, ak existuje $f \in \mathsf{E}$, ktorá potrebuje obvod veľkosti $2^{\gamma n}$ (pre $\gamma > 0$), tak máme úplnú derandomizáciu $\mathsf{BPP} = \mathsf{P}$.

Skrátka, ak existuje $f \in \mathsf{E}$, ktorá potrebuje

$$\text{obvod veľkosti } \begin{Bmatrix} - \\ n^{\omega(1)} \\ 2^{n^\varepsilon} \\ 2^{\Omega(n)} \end{Bmatrix} \text{ tak } \begin{Bmatrix} \mathsf{BPP} & \subseteq & \mathsf{EXP} \\ \mathsf{BPP} & \subseteq & \mathsf{SUBEXP} \\ \mathsf{BPP} & \subseteq & \mathsf{QuasiP} \\ \mathsf{BPP} & = & \mathsf{P} \end{Bmatrix}.$$

Čím ťažšie funkcie existujú, tým lepšie vieme derandomizovať.

Kapitola 17

Pseudonáhodné generátory

Obr. 17.1: Pomerne slabý pseudonáhodný generátor.

17.1 Derandomizácia pomocou pseudonáhodných generátorov

Ako zistíme, či je pseudonáhodný generátor (PNG) „dobrý“? Ak za mnou niekto príde, že navrhol nový generátor, ako zistím, či produkuje „dostatočne náhodné“ čísla?

V praxi sa na testovanie používajú batérie štatistických testov, napríklad Diehard (Marsaglia, 1996), TestU01 (L'Ecuyer a Simard, 2007), NIST (Rukhin a spol., 2001). Každý test väčšinou spočíta nejakú hodnotu, ktorej distribúciu poznáme a ak je táto hodnota len veľmi málo pravdepodobná, prehlási, že vstup nie je náhodný. Napríklad:

- Frekvenčný test: počet núl a jednotiek (n_0, n_1) by mal byť zhruba rovnaký, teda ak je rozdiel $n_1 - n_0$ príliš veľký[1], prehlásime, že postupnosť nie je náhodná.

[1] Čo je to príliš veľký? Počet jednotiek n_1 má binomické rozdelenie, resp. $Z = (n_1 - n_0)/\sqrt{n}$ má približne normálne rozdelenie a $\Pr(Z \leq z) \approx \Phi(z)$, teda $\Pr(|Z| \geq z) \approx 2(1 - \Phi(z))$ – ak je táto hodnota povedzme 1%, znamená to, že pre skutočne náhodné čísla by takáto situácia nastala iba v 1% prípadov, čo je dosť málo pravdepodobné.

- Všobecnejší blokový frekvenčný test: rozdeľme postupnosť dĺžky n na k blokov dĺžky m; ak spočítame podiel jednotiek p_i v každom bloku, dostaneme distribúciu hodnôt, ktorá by sa mala podobať na binomické (v limite normálne) rozdelenie. A ako zistíme, či sa dve distribúcie dostatočne podobajú? Na to sa používa tzv. test dobrej zhody (goodness of fit); jeden zo spôsobov je tzv. chí-kvadrát test, kde spočítame súčet štvorcov odchýliek $X = \sum(p_i - o_i)^2/o_i$, kde p_i sú pozorované a o_i očakávané hodnoty; premenná X má približne χ^2_{k-1} rozdelenie s $k-1$ stupňami voľnosti[2]; ak je hodnota X príliš vysoká, generátor odmietneme.

- Test behov a medzier: spočítajme distribúciu dĺžok súvislých úsekov núl a jednotiek (dĺžky tzv. medzier a behov); očakávame, že behov/medzier dĺžky k je zhruba $1/2^k$; pozorované distribúcie porovnáme s teoretickými pomocou χ^2-testu.

- Narodeninový test: vyberieme m dátumov narodenín z roku, ktorý má n dní, zotriedime, zrátame počet dní medzi narodeninami a spočítame, koľko medzier sa opakuje viackrát; toto číslo by malo mať Poissonovu distribúciu s parametrom $\lambda = m^3/4n$. Napríklad pre $n = 2^{32}$, $m = 2^{12}$, $\lambda = 4$; pokus zopakujeme 5000-krát a spočítame χ^2-test.

- Gorilí test: vygenerujeme $2^{26} + 25$ bitov, nech x je počet 26-bitových slov, ktoré sa v tejto postupnosti *nenachádzajú*; potom x by malo mať zhruba normálnu distribúciu s priemerom 24687971 a štandardnou deviáciou 4170, takže $\Phi((x - 24687971)/4170)$ by malo byť rovnomerne náhodné číslo medzi 0 a 1.

- Test hodností matíc: rozdeľme postupnosť na bloky dĺžky $k \times \ell$, kde každý blok predstavuje jednu binárnu maticu; spočítajme hodnosti (ranky) týchto matíc a porovnajme s teoretickým rozdelením pomocou χ^2-testu.

- Gcd test: spustíme Euklidov algoritmus na náhodných 32-bitových vstupoch a spočítame 1) k – počet iterácií a 2) konečnú hodnotu gcd. Ak spustíme tento test 10-milión-krát, očakávané počty rôznych k a rôznych gcd by mali byť zhruba ako v tabuľke (platí: $\Pr[\gcd = x] = \frac{6}{\pi^2 x^2}$):

k	≤ 3	4	5	6	7	8	9	10	$\cdots$
očak. počet:	5.5	29.5	144.6	590.7	2065	6277	16797	39965	
gcd	1	2	3	4	5	6	7	8	$\cdots$
očak. počet:	6079271	1519817	675474	379954	243171	168869	124067	94989	

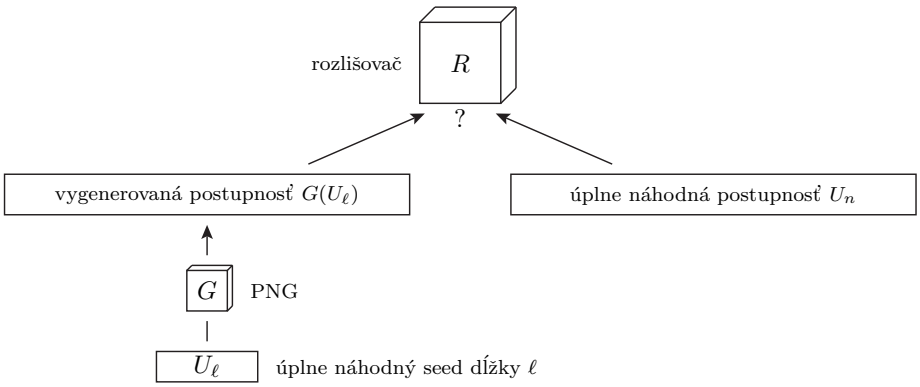
  Opäť použijeme χ^2 na test dobrej zhody, či pozorované dáta zodpovedajú očakávanému rozdeleniu.

Vo všeobecnosti je štatistický test nejaký algoritmus, ktorý na vstupe dostane postupnosť bitov, niečo si popočíta a nakoniec rozhodne, či vstup vyzerá náhodne, alebo nie. Väčšinou si spočíta nejakú štatistiku – nejakú hodnotu Y,

[2]Distribúcia χ^2_k je súčet k normálnych distribúcií $\mathrm{N}(0,1)$.

ktorej distribúciu poznáme (za predpokladu nulovej hypotézy, že vstup je postupnosť nezávislých náhodných bitov) a vyhlási, že postupnosť je nenáhodná, ak je spočítaná štatistika príliš malá alebo naopak príliš veľká.

Štatistický test sa teda snaží *rozlíšiť* náhodné postupnosti od tých, ktoré nevyzerajú náhodne. Takémuto algoritmu budeme tiež hovoriť „rozlišovač“. Ako sme spomínali, v praxi sa pri testovaní pseudonáhodných generátorov používajú celé batérie štatistických testov. V teórii si môžeme dovoliť definíciu ešte sprísniť: budeme hovoriť, že PNG je dobrý vtedy, ak prejde cez *všetky efektívne* štatistické testy. Alebo inými slovami, PNG je dobrý, ak *žiadny efektívny* algoritmus nedokáže rozlíšiť distribúciu ním vygenerovaných pseudonáhodných postupností od skutočne náhodnej distribúcie. Poďme si tieto pojmy zadefinovať exaktnejšie:



Obr. 17.2: Pseudonáhodný generátor G dostane na vstupe seed dĺžky ℓ a vygeneruje pseudonáhodnú postupnosť dĺžky m. Dobrý PNG je taký, že výslednú distribúciu reťazcov je ťažké odlíšiť od skutočne náhodných. Presnejšie: Ľubovoľný (dostatočne malý) obvod, rozlišovač R, si „nevšimne“ rozdiel medzi $G(U_\ell)$ a uniformnou distribúciou U_m – pravdepodobnosť, že C dá na týchto vstupoch iné výsledky je malá.

Definícia 17.1 (Rozlišovač a pseudonáhodná distribúcia). *Nech D je distribúcia nad $\{0,1\}^m$. Hovoríme, že obvod R je rozlišovač pre D, ak*

$$\left| \Pr_{x \in_R D}[R(x) = 1] - \Pr_{r \in_R U_m}[R(r) = 1] \right| > 1/10,$$

kde U_m je uniformná distribúcia na m bitoch. Skrátene budeme pravdepodobnosť, že R akceptuje na vstupe z distribúcie D značiť $\Pr[R(D)] = \Pr_{x \in_R D}[R(x) = 1]$.

Hovoríme, že distribúcia je S-pseudonáhodná, ak neexistuje rozlišovač veľkosti S. To znamená že pre každý obvod C veľkosti S je

$$\Pr[C(D)] \approx \Pr[C(U_m)] \pm 1/10.$$

Definícia 17.2 (Pseudonáhodný generátor). *$S(\ell)$-pseudonáhodný generátor*[3]

[3] $S : \mathbb{N} \to \mathbb{N}$ je časovo konštruovateľná, neklesajúca

je

- *funkcia* $G : \{0,1\}^* \to \{0,1\}^*$*, ktorá na vstupe dĺžky* ℓ *vyprodukuje v čase* $O(2^\ell)$ *postupnosť dĺžky* $S(\ell)$ *a*
- $G(U_\ell)$ *je* $S(\ell)^3$*-pseudonáhodná distribúcia.*

Vstup, teda inicializačnú hodnotu generátora, voláme tiež seed a $S(\ell)$ *je predĺženie. Napríklad* $2^{\Omega(\ell)}$*-PNG znamená, že na postupnosť dĺžky* m *nám stačí seed dĺžky* $\ell = O(\log m)$ *a neexistuje obvod veľkosti* m^3*, ktorý by pseudonáhodnú postupnosť odlíšil od náhodnej.*

Všimnime si, že PNG má oveľa viac času ako obvody, ktoré sa ho snažia rozlíšiť! Nám by napríklad stačil aj PNG, ktorý v čase $O(2^\ell)$ vygeneruje postupnosť dĺžky $m = 2^{0.001\ell}$. Takýto generátor pracuje v čase $O(m^{1000})$ (v závislosti od dĺžky výstupu), zatiaľčo rozlišovače sú iba obvody veľkosti m^3. Preto hypotéza, že dobré PNG existujú, znie celkom uveriteľne.[4]

Poďme si teraz ukázať, že ak existuje dobrý PNG G, vieme vďaka nemu derandomizovať ľubovoľný BPP-algoritmus A. Hlavná myšlienka je jednoduchá: namiesto náhodných bitov použijeme tie pseudonáhodné. (Koniec-koncov, v praxi sa často pravdepodobnostné algoritmy spúšťajú len s nejakým štandardným knižničným PNG – bez dôkazu, že to funguje.) Ak však použijeme *dobrý* PNG a vyskúšame *všetky* možné seedy s, dokážeme spočítať $\Pr[A(x, G(U_\ell)]$, čo je $\Pr[A(x, U_m)] \pm 10\%$ a vezmeme väčšinovú odpoveď. Keďže BPP-algoritmy akceptujú s pravdepodobnosťou buď $\geq 2/3 > 66\%$ alebo $\leq 1/3 < 34\%$, nejakých $\pm 10\%$ výsledok neovplyvní.

Prečo to funguje? Nuž ak by neplatilo $\Pr[A(x, G(U_\ell)] = \Pr[A(x, U_m)] \pm 10\%$ (pre všetky x od určitej veľkosti), tak algoritmus A nám poslúži ako rozlišovač! V takom prípade existuje nekonečná postupnosť „zlých“ vstupov x, na ktorých sa $\Pr[A(x, G(U_\ell)]$ a $\Pr[A(x, U_m)]$ dosť líšia. Tieto vstupy môžeme zadrátovať do obvodu pre A. Napríklad v Miller-Rabinovom teste prvočíselnosti (pozri sekciu 2.6) namiesto náhodných $0 < a < n$ vygenerujeme pseudonáhodné (pre všetky možné seedy). Ak by takýto test nefungoval, potom Miller-Rabinov test (na konkrétnych zložených číslach) dokáže rozlíšiť vygenerované pseudonáhodné od skutočne náhodných bitov.

Lema 17.1. *Ak existuje* $2^{\Omega(\ell)}$*-PNG, potom* BPP = P.

■ **Dôkaz.** Majme $2^{\varepsilon\ell}$-PNG G a BPP-algoritmus A, ktorý pracuje v čase $m = O(n^k)$ (takže používa najviac m náhodných bitov). Nahraďme v algoritme náhodné bity pseudonáhodnými. Keďže G je dobrý PNG, pre všetky (až

[4] Len tak na okraj, kvôli perspektíve dodajme, že predpoklad, že existuje dobrý PNG je ešte veľmi skromný napríklad v porovnaní s tým, že existujú *kryptograficky silné* PNG – čomu veríme tiež. Pri kryptograficky silných PNG nám síce stačí len polynomiálne predĺženie, avšak rozlišovače môžu pracovať v ľubovoľne veľkom polynomiálnom čase a napriek tomu vyžadujeme, aby bol rozdiel $|\Pr[R(G(U_\ell))] - \Pr[R(U_m)]|$ zanedbateľný (tzn. $1/m^{\omega(1)}$ – menší ako $1/m^c$ pre ľubovoľne veľké c pre dostatočne veľké m). V definícii kryptograficky silných PNG sú rozlišovače silnejšie (majú viac času) ako samotný generátor (čo dáva zmysel, ak chceme byť odolní voči útoku výpočtovo silných protivníkov, akými sú nadnárodné korporácie, či štátni aktéri).

na konečne veľa) x platí:

$$\Pr[A(x, G(U_\ell))] \approx \Pr[A(x, U_m)] \pm \frac{1}{10}.$$

Teda rozdiel medzi pravdepodobnosťou akceptovania pri náhodných a pseudonáhodných bitoch je len 1/10. Ak by sa totiž algoritmus A nesprával s PNG podobne ako s naozaj náhodnými bitmi, mohli by sme ho využiť ako „test“, ktorý rozlišuje daný PNG G od skutočne náhodných bitov: Obvod, ktorý má zadrátované zlé vstupy x a počíta funkciu $r \mapsto A(x, r)$ je obvod, ktorý rozoznáva $G(U_\ell)$ od U_m, čo je spor.

Pripomeňme, že BPP-algoritmus pre $x \in L(A)$ akceptuje s pravdepodobnosťou aspoň 2/3 a pre $x \notin L(A)$ najviac 1/3. Ak máme dobrý PNG, pravdepodobnosť bude ±rovnaká a navyše $\Pr[A(x, G(U_\ell))]$ vieme vypočítať úplne presne tak, že vyskúšame všetky seedy:

Na derandomizáciu stačí algoritmus A spustiť pre všetky seedy dĺžky $\ell = \frac{1}{\varepsilon} \log m = O(\log n)$ a rozhodnúť sa podľa väčšinovej odpovede. Ak $x \in L(A)$, pravdepodobnosť akceptovania so skutočne náhodnými bitmi je $> \frac{2}{3}$ a s PNG je to najviac o $\frac{1}{10}$ menej, čo je však stále viac ako polovica a na základe väčšinovej odpovede algoritmus povie Áno. Naopak, ak $x \notin L(A)$, tak $\Pr[A(x, U_m)] < \frac{1}{3}$ a s PNG to je maximálne o $\frac{1}{10}$ viac, čo je stále menej ako polovica a teda väčšinová odpoveď je Nie.

$$x \in L(A) \Longrightarrow \Pr[A(x, G(U_\ell)] = \Pr[A(x, U_m)] \pm 0.1 > 2/3 - 0.1 > 0.5$$
$$x \notin L(A) \Longrightarrow \Pr[A(x, G(U_\ell)] = \Pr[A(x, U_m)] \pm 0.1 < 1/3 + 0.1 < 0.5$$

□

Túto lemu a dôkaz môžeme zovšeobeniť: ak existuje $S(\ell)$-PNG a my budeme skúšať všetky seedy dĺžky $\ell(n)$, tak pravdepodobnostné algoritmy bežiace v čase $S(\ell(n))$ vieme derandomizovať v čase $2^{O(\ell(n))}$.

Zovšeobecnenie lemy 17.1. *Ak existuje $S(\ell)$-PNG (kde $\ell(n)$ je vypočítateľná v polynomiálnom čase), tak*

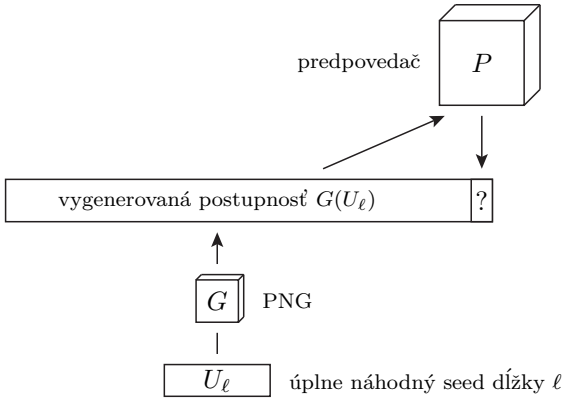
$$\mathsf{BPTIME}(S(\ell(n))) \subseteq \mathsf{DTIME}(2^{O(\ell(n))}).$$

Konkrétne napríklad ak existuje

- $2^{\ell^{\Omega(1)}}$*-PNG, tak* $\ell = \log^{O(1)} n$*, takže* $\mathsf{BPP} \subseteq \mathsf{QuasiP} = \mathsf{DTIME}(2^{\mathsf{polylog}(n)})$*;*
- $\ell^{\omega(1)}$*-PNG, tak* $\ell = n^{o(1)}$*, takže* $\mathsf{BPP} \subseteq \mathsf{SUBEXP} = \bigcap_{\varepsilon>0} \mathsf{DTIME}(2^{n^\varepsilon})$.

17.2 Rozlišovače a predpovedače

Dobrý pseudonáhodný generátor sme zadefinovali ako taký, ktorého distribúciu nevieme efektívne *rozlíšiť* od skutočne náhodnej. Ďalej sa nám však bude hodiť iný pohľad na dobré PNG: výstup z dobrého PNG by mal byť *nepredvídateľný* – ak dostaneme len časť výstupu, nevieme efektívne *predpovedať nasledujúci bit* (pozri obrázok 17.3).



Obr. 17.3: Pseudonáhodný generátor G dostane na vstupe seed dĺžky ℓ a vygeneruje pseudonáhodnú postupnosť. Dobrý PNG je taký, že výslednú distribúciu reťazcov je ťažké predpovedať. Presnejšie: Ľubovoľný (dostatočne malý) obvod, predpodovedač P, nedokáže predpovedať i-ty bit z predošlých – pravdepodobnosť, že sa trafí, je len o kúsok viac ako 50%.

Definícia 17.3 (Predpovedač). *Nech D je distribúcia nad $\{0,1\}^m$. Hovoríme, že P je predpovedač pre D, ak pre nejaké i dokáže P predpovedať i-ty bit z predošlých s výrazne väčšou úspešnosťou ako len náhodné tipovanie. Konkrétne budeme vyžadovať, že P predpovedá i-ty bit správne s pravdepodobnosťou aspoň $50\% + 1/10m$:*

$$\Pr_{r\in_R D}[P(r_1,\ldots,r_{i-1}) = r_i] > \frac{1}{2} + \frac{1}{10m}.$$

Všimnite si, že nám stačí menšia úspešnosť $(1/10m)$ ako pri rozlišovačoch $(1/10)$. Ukážeme si, že ak je distribúcia nepredvídateľná, tak je nerozlíšiteľná od skutočne náhodnej:

Veta 17.1. *Ak existuje rozlišovač veľkosti S pre D, potom existuje aj predpovedač veľkosti $2S$.*

■ **Dôkaz.** Nech R je daný rozlišovač. Predpovedač P na vstupe $i, y_1, \ldots, y_{i-1}$ doplní postupnosť o náhodné $z_i, \ldots, z_m$ a spustí rozlišovač: nech

$$a = R(y_1, \ldots, y_{i-1}, z_i, \ldots, z_m).$$

Ak $a = 1$ (postupnosť vyzerá nenáhodne), predpovedač bude predpokladať, že tip z_i bol správny a odpovie z_i; v opačnom prípade odpovie $1 - z_i$. Takýto predpovedač síce používa náhodné bity, ale tie sa dajú vymeniť za neuniformnosť: ak funguje v priemernom prípade, potom existuje voľba konktrétnych postupností $z_i, \ldots, z_m$, pre ktoré bude predpovedač fungovať.

Dôkaz, že takýto predpovedač bude úspešný využíva tzv. „hybridný argument“: predstavme si $m+1$ distribúcií $D_0, \ldots, D_m$, kde D_0 je úplne rovnomerne

náhodná distribúcia U_m, $D_m = D$ a distribúcie medzi tým tvoria „plynulý predchod“ od U_m ku D, konkrétne D_i má prvých i bitov z D a zvyšok sú úplne náhodné bity.

Označme $p_i = \Pr[R(D_i)]$. Keďže R je rozlišovač, $p_m - p_0 = \Pr[R(D)] - \Pr[R(U_m)] \geq \frac{1}{10}$. Ak rozpíšeme $p_m - p_0$ ako teleskopickú sumu $\sum p_i - p_{i-1} = p_m - p_0$, musí existovať i, pre ktoré je $p_i - p_{i-1} \geq \frac{1}{10m}$. Tento i-ty bit budeme vedieť úspešne predpovedať:

P predpovie i-ty bit správne buď ak $y_i = z_i$ a R povie Áno, alebo ak $y_i = 1 - z_i$ a R povie Nie. Uvažujme všetky 4 možnosti, ktoré môžu nastať: či je $z_i \stackrel{?}{=} y_i$ a či R na vstupe $(y_1, \ldots, y_{i-1}, z_i, \ldots, z_m)$ odpovie Áno alebo Nie:

	Áno	Nie	spolu
$z_i = y_i$	A	B	1/2
$z_i \neq y_i$	C	D	1/2
spolu	p_{i-1}	$1 - p_{i-1}$	1

Ak označíme pravdepodobnosti jednotlivých udalostí A, B, C, D ako v tabuľke, potom nás zaujíma hodnota A + D.

Keďže z_i je náhodný bit, oba prípady, $z_i = y_i$ a $z_i \neq y_i$ nastanú s rovnakou pravdepodobnosťou 1/2. Ďalej vieme, že R odpovie Áno s pravdepodobnosťou p_{i-1}; avšak za predpokladu, že $z_i = y_i$ dostaneme distribúciu D_i a R odpovie Áno s pravdepodobnosťou p_i. Takže pravdepodobnosť

$$\begin{aligned} \mathrm{A} &= \Pr[\text{Áno} \mid z_i = y_i] \cdot \Pr[z_i = y_i] = \Pr[R(D_i)] \cdot \tfrac{1}{2} = \tfrac{1}{2} p_i \\ \mathrm{A} + \mathrm{C} &= \Pr[\text{Áno}] = \Pr[R(D_i)] = p_{i-1} \\ \mathrm{C} + \mathrm{D} &= \Pr[z_i \neq y_i] = 1/2 \end{aligned}$$

Z toho pravdepodobnosť, že P správne predpovie i-ty bit je

$$\mathrm{A} + \mathrm{D} = \mathrm{C} + \mathrm{D} - (\mathrm{A} + \mathrm{C}) + 2 \times \mathrm{A} = \frac{1}{2} + p_i - p_{i-1}. \qquad \square$$

Dôsledok 17.1. *Funkcia $G : \{0,1\}^* \to \{0,1\}^*$ s predĺžením $S(\ell)$ vypočítateľná v $O(2^\ell)$ je $S(\ell)$-pseudonáhodný generátor, ak pre ňu neexistuje predpovedač veľkosti $2S(\ell)^3$.*

17.3 Ako vyrobiť pseudonáhodný generátor

Ako sme spomínali v úvode k tejto časti, dobrý pseudonáhodný generátor vytvoríme z pekelne ťažkej funkcie – tú vytvoríme z veľmi ťažkej funkcie – a tú vytvoríme z ťažkej funkcie. Nastal čas presne si zadefinovať, čo týmito pojmami myslíme.

Keď budeme hovoriť o ťažkosti v priemernom prípade, môžeme hovoriť o počte alebo o časti vstupov, na ktorých dá obvod správnu odpoveď. Avšak prirodzené je použiť v tomto prípade jazyk pravdepodobnosti: Ak obvod odpovedá správne na 99% vstupov, znamená to, že keby sme vygenerovali úplne

náhodný vstup (rovnomerne z $\{0,1\}^n$), tak pravdepodobnosť správnej odpovede je 0.99. Budeme tiež hovoriť, že obvod C aproximuje funkciu f na 99%, ak $\Pr_{x\in_R\{0,1\}^n}[f(x) = C(x)] \geq 0.99$.

Definícia 17.4 (Ťažká, veľmi ťažká a pekelne ťažká funkcia). *Budeme hovoriť, že funkcia f je*

- *ťažká, ak ju obvody veľkosti $S \leq 2^{\gamma n}$ nedokážu spočítať presne (pre nejakú konštantu $\gamma > 0$),*
- *veľmi ťažká, ak ju obvody veľkosti $S \leq 2^{\gamma n}$ nedokážu ani len aproximovať na 99% (pre nejakú konštantu $\gamma > 0$),*
- *pekelne ťažká, ak ju obvody veľkosti $S \leq 2^{\gamma n}$ nedokážu ani len aproximovať na $1/2 + 1/S$ vstupoch (pre nejakú konštantu $\gamma > 0$).*

Všeobecnejšie by sme mohli definovať

Definícia 17.5 (Ťažkosť v priemernom prípade). *Pre $f_n : \{0,1\}^n \to \{0,1\}$ a $\rho \in [0,1]$ definujeme ťažkosť $\mathsf{H}^\rho_{\mathrm{avg}}(f_n) = S$, ak žiadny obvod veľkosti S nedokáže ρ-aproximovať f_n. Teda*

$$\mathsf{H}^\rho_{\mathrm{avg}}(f_n) = \max\{S \mid \forall \textit{obvod } C,\ |C| \leq S : \Pr_{x\in_R\{0,1\}^n}[C(x) = f_n(x)] < \rho\}.$$

Pre $f : \{0,1\}^* \to \{0,1\}$, t.j. pre ľubovoľnú veľkosť vstupu definujeme $\mathsf{H}^\rho_{\mathrm{avg}}(f)$ prirodzene tak, že uvažujeme rôzne n samostatne: označme $f_n = f|_{\Sigma^n}$ zúženie funkcie na vstupy dĺžky n; funkciu „pozliepame“ z $\mathsf{H}^\rho_{\mathrm{avg}}(f)(n) = \mathsf{H}^\rho_{\mathrm{avg}}(f_n)$ pre všetky n.

Zložitosť v najhoršom prípade je $\mathsf{H}_{\mathrm{wrs}}(f) = \mathsf{H}^1_{\mathrm{avg}}(f)$ (f chceme spočítať na 100%; rozmyslite si, že $\mathsf{H}_{\mathrm{wrs}}(f) = S$ znamená, že $f \notin \mathsf{SIZE}(S)$). Priemernú zložitosť definujeme ako $\mathsf{H}_{\mathrm{avg}}(f) = \max\{S \mid \mathsf{H}^{1/2+1/S}_{\mathrm{avg}}(f) \geq S\}$.

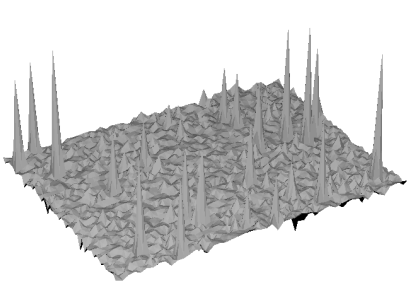
Potom f je

- *ťažká*, ak $\mathsf{H}_{\mathrm{wrs}}(f) = 2^{\Omega(n)}$,
- *veľmi ťažká*, ak $\mathsf{H}^{0.99}_{\mathrm{avg}}(f) = 2^{\Omega(n)}$,
- *pekelne ťažká*, ak $\mathsf{H}_{\mathrm{avg}}(f) = 2^{\Omega(n)}$.

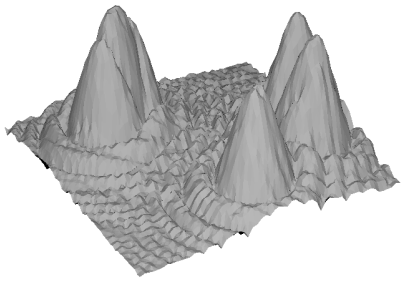
Geometrický pohľad (pozri obrázok 17.4): Môžeme si skúsiť predstaviť priestor, kde každý bod je jedna funkcia (každá funkcia je vlastne vektor 2^n hodnôt). Hodnota

$$\Delta(f,g) = \Pr_x[f(x) \neq g(x)]$$

definuje vzdialenosť dvoch funkcií (nie je to nič iné ako normalizovaná Hammingova vzdialenosť tabuliek hodnôt) a g aproximuje f na 99%, ak je vzdialenosť f a g najviac 0.01. Každému bodu takéhoto priestoru priradíme hodnotu $\mathsf{H}_{\mathrm{wrs}}(g)$



(a) Špicaté kopce: niektoré funkcie sú síce veľmi ťažké, ale neďaleko nich je vždy nejaká ľahká funkcia.



(b) Veľké kopce: existujú funkcie, ktoré sa nedajú spočítať ani približne: aj v blízkom okolí sú funkcie ťažké.

Obr. 17.4: Predstavme si, že rovina xy znázorňuje všetky funkcie, pričom body, ktoré sú blízko sú podobné funkcie, tzn. na veľa vstupoch dávajú rovnaký výsledok. Predstavme si, že výška znázorňuje zložitosť (veľkosť najmenšieho obvodu, ktorý funkciu počíta).

– veľkosť najmenšieho obvodu, ktorý počíta danú funkciu. Ťažká funkcia má vysokú hodnotu, veľmi ťažká má vysoké hodnoty všade vo vzdialenosti do 0.01.

$$\mathsf{H}^{1-\delta}_{\text{avg}}(f) = \max\{\mathsf{H}_{\text{wrs}}(g) \mid \Delta(f,g) \leq \delta\}.$$

Poďme sa teraz pustiť do konštrukcie dobrého pseudonáhodného generátora. Predpokladajme, že existuje pekelne ťažká funkcia.

Rozcvička 1: Ako rozcvičku skúsme najskôr vytvoriť PNG, ktorý vstup predĺži o 1 bit.

To je triviálne, stačí zobrať $G(z) = z\, f(z)$. Zjavne z sú náhodné bity, ktoré sa nedajú predpovedať vôbec a ani $f(z)$ sa nedá ľahko predpovedať zo z. Dôkaz: Predpovedač posledného bitu P je (malý) obvod, ktorý aproximuje $f(z)$ na aspoň $1/2+1/10m$ vstupoch, ale žiadny (malý) obvod C nedokáže aproximovať f na viac ako $1/2 + 1/S$, pretože f je pekelne ťažká funkcia:

$$\exists P, |P| \leq 2m^3 : \Pr_{z\in_r\{0,1\}^\ell}[P(z) = f(z)] \geq 1/2 + 1/10m \quad \textit{(predpovedač)}$$

$$\forall C, |C| \leq S : \Pr_{z\in_r\{0,1\}^\ell}[C(z) = f(z)] \leq \frac{1}{2} + 1/S \quad \textit{(pekelne ťažká funkcia)}$$

a to je spor už pre $S(\ell) \geq 2m^3 = 2(\ell+1)^3$.

Rozcvička 2: Skúsme vytvoriť PNG, ktorý predlžuje o 2 bity.

Jednoducho rozdelíme vstupný reťazec z na dve polovice z_1, z_2 a definujme $G(z) = z_1\, f(z_1)\, z_2\, f(z_2)$. Zrejme z_1, z_2 sa nedá predpovedať a $f(z_1)$ zdôvodníme rovnako, ako vyššie. Ale čo $f(z_2)$? Nemohla by nám vedomosť $f(z_1)$ pomôcť pri

predpovedaní $f(z_2)$? Všeobecnejšie: ak by sme z rozdelili na k častí a definovali $G(z) = z_1\, f(z_1)\, z_2\, f(z_2) \cdots z_k\, f(z_k)$, mohla by nám vedomosť hodnôt f na $k-1$ náhodných vstupoch pomôcť pri predpovedaní $f(z_k)$?

Odpoveď je Nie. Pri dôkaze použijeme priemerovací argument: ak je priemer sady čísel väčší ako t, potom nejaké konkrétne číslo z tejto sady musí byť väčšie ako t. V našom prípade budeme používať nasledovnú formu: ak $\Pr_{x\in_R X, y\in_R Y}[E_{x,y}] > t$, čo chápeme ako priemer cez všetky x, tak existuje konkrétne $x \in X$, pre ktoré je $\Pr_{y\in_R Y}[E_{x,y}] > t$.

Ak by existoval predpovedač P taký, že

$$\Pr_{z_1,z_2\in_R\{0,1\}^{\ell/2}}[P(z_1\, f(z_1)\, z_2) = f(z_2)] > \frac{1}{2} + \frac{1}{10m},$$

potom existuje aspoň jedno konkrétne z_1 také, že

$$\exists z_1 \in \{0,1\}^{\ell/2} : \Pr_{z_2\in_R\{0,1\}^{\ell/2}}[P(z_1\, f(z_1)\, z_2) = f(z_2)] > \frac{1}{2} + \frac{1}{10m},$$

Toto z_1 spolu s hodnotou $f(z_1)$ môžeme „zadrátovať“ do obvodu P a dostaneme zhruba rovnako veľký obvod P'

$$P'(x) = P(\underbrace{z_1\, f(z_1)}_{\text{zadrátovaná konštanta}}\, x),$$

ktorý počíta funkciu f na veľkej časti vstupov:

$$\Pr_{x\in_R\{0,1\}^{\ell/2}}[P'(x) = f(x)] > \frac{1}{2} + \frac{1}{10m},$$

čo je spor s tým, že f je pekelne ťažká (už pre $S(\ell/2) \geq 2(\ell+2)^3$).

Problém je, že takýmto prístupom, že seed rozdelíme na niekoľko disjunktných častí, ktoré vopcháme do f sa nedostaneme cez predĺženie 2ℓ. Prirodzený ďalší nápad je použiť podpostupnosti z, pričom niektoré bity použijeme do viacerých vstupov. Ak $I = \{i_1, \ldots, i_k\}$ je podmnožina $\{1, 2, \ldots, \ell\}$, označme $z_I = z_{\{i_1,\ldots,i_k\}} = z_{i_1} \cdots z_{i_k}$. Vo všeobecnosti pre sadu takýchto podmnožín definujeme generátor nasledovne:

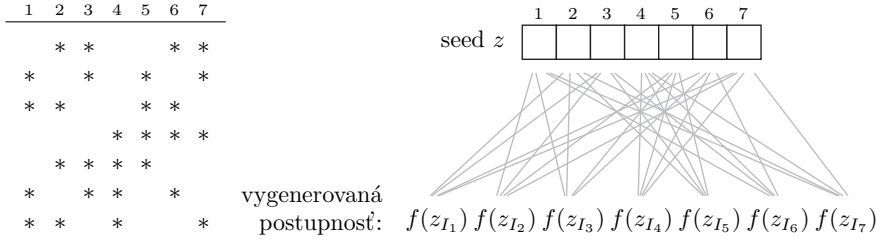
Definícia 17.6 (Nisan-Wigdersonov generátor). *Nech $f : \{0,1\}^k \to \{0,1\}$ a $\mathcal{I} = \{I_1, \ldots, I_m\}$ je sada k-prvkových podmnožín $\{1, \ldots, \ell\}$. Potom definujeme Nisan-Wigdersonov generátor ako funkciu $NW^f_{\mathcal{I}} : \{0,1\}^\ell \to \{0,1\}^m$,*

$$NW^f_{\mathcal{I}}(z) = f(z_{I_1})\, f(z_{I_2}) \cdots f(z_{I_m}),$$

kde z_I je podreťazec z s indexmi z I. Pozri príklad na obrázku 17.5b.

Samozrejme, ak majú dve podmnožiny I_i, I_j veľký prienik, znalosť hodnoty $f(z_{I_i})$ by mohla pomôcť predpovedať hodnotu $f(z_{I_j})$. My si ukážeme, že

1. existujú veľmi veľké sady množín s malými prienikmi a



Obr. 17.5: Príklad Nisan-Wigdersonovho generátora pre sadu množín $\mathcal{I} = \{\{2,3,6,7\},\{1,3,5,7\},\{1,2,5,6\},\{4,5,6,7\},\{2,3,4,5\},\{1,3,4,6\},\{1,2,4,7\}\}$ zobrazenú vľavo: každý riadok predstavuje jednu množinu veľkosti 4; každé dva riadky majú 2 spoločné prvky (je to $(7,4,2)$-dizajn). Samozrejme, tento konkrétny generátor je nanič, pretože vôbec nepredlžuje – na dobrý PNG potrebujeme exponenciálne veľkú sadu množín.

2. $f(z_{I_j})$ sa nedá ľahko predpovedať ani z veľa hodnôt $f(z_{I_i})$, ak majú množiny I_i malý prienik s I_j.

Prvý bod je doménou matematickej oblasti kombinatorických štruktúr:

Definícia 17.7 (Kombinatorický dizajn). *Sadu množín $\mathcal{I} = \{I_1, \ldots, I_m\}$ voláme (ℓ, k, d)-dizajn pre $\ell > k > d$, ak*

- *všetky množiny I_j sú podmnožiny $\{1, 2, \ldots, \ell\}$,*
- *všetky množiny majú veľkosť $|I_j| = k$,*
- *každé dve množiny majú prienik veľkosti $|I_i \cap I_j| \leq d$ $(\forall i \neq j)$.*

(Presnejšie ide o tzv. packing design – v klasickej definícii dizajnu, v slovenskej literatúre tiež (ℓ, k, d)-konfigurácie, majú prieniky veľkosť presne $|I_i \cap I_j| = d$.)

Príklad malého pakovacieho dizajnu je na obrázku 17.6.

Ak trochu predbehneme, z dôkazov vyplynú rôzne obmedzenia na voľbu parametrov. My budeme chcieť konkrétne pre funkciu s $\mathsf{H}_{\mathrm{avg}}(f_k) \geq 2^{\gamma k}$ dizajn s parametrami

- m – exponenciálny počet množín $2^{d/10} = 2^{\Omega(\ell)}$ – toto zodpovedá dĺžke vygenerovanej postupnosti,
- ℓ – seed dĺžky $c \cdot \log m$, kde $c = (20/\gamma)^2$,
- k – veľkosť množín $\sqrt{c} \cdot \log m$ – toto bude dĺžka vstupu pre f_k,
- d – veľkosť prieniku $10 \cdot \log m$

Obr. 17.6: $(24, 8, 4)$-dizajn: 759 stĺpcov, každý stĺpec predstavuje jednu množinu veľkosti 8; dve množiny majú prienik veľkosti ≤ 4. Mimochodom, autor ho našiel jednoduchým greedy algoritmom, následne zistil, že ide o známy Steinerovský systém $S(5, 8, 24)$, známy aj ako Wittov dizajn (každá 5-prvková množina sa vyskytuje práve v jednom stĺpci; $759 \times \binom{8}{5} = \binom{24}{5}$) a množiny, čítané ako vektory $\mathbb{Z}_2^{24}$ sú práve vektory váhy 8 z tzv. Golayovho kódu. V teórii kódovania nás tiež zaujímajú sady vektorov, ktoré sú ďaleko od seba (pri Hammingovej vzdialenosti), pretože ak pošleme správu zloženú z takýchto vektorov, vieme ju zrekonštruovať aj v prípade, že sa počas prenosu zopár bitov preklopí. Pakovacie dizajny sú presne kódy s konštantnou váhou a vzdialenosťou $2d$.

Seed logaritmickej dĺžky potrebujeme, ak chceme derandomizovať v polynomiálnom čase. Pri konštrukcii dizajnu zase budeme potrebovať, aby $\ell \geq 10k^2/d$ a d bolo aspoň $10 \log m$. Z toho vyplýva, že $k \leq \sqrt{c} \cdot \log m$. Pri dôkaze kvality PNG budeme potrebovať, aby $2d \leq \gamma k$; z toho plynie voľba konštanty $c = (20/\gamma)^2$.

Mimochodom, konštanty ktoré takto vzniknú budú astronomické a absolútne nepraktické. Ak napríklad predpokladáme, že existuje funkcia $f \in \mathsf{E}$ s $\gamma = 0.1$, tak $k = 200 \log m$ a $\ell = 40\,000 \log m$. Deterministický algoritmus potom skúša všetkých $2^\ell = m^{40\,000}$ možností. Doron a spol. (2019) dokázali, že zo silnejších predpokladov sa dajú pravdepodobnostné algoritmy bežiace v čase t derandomizovať v čase $t^{2+O(\varepsilon)}$.

Lema 17.2 (Nisan a Wigderson (1994)). *Nech $\mathcal{I}$ je (ℓ, k, d)-dizajn veľkosti m s parametrami vyššie a $f : \{0,1\}^k \to \{0,1\}$ je pekelne ťažká funkcia. Potom $NW_{\mathcal{I}}^f$ je $2^{\Omega(\ell)}$-PNG.*

Dôkaz využíva podobnú myšlienku ako rozcvička 2: ak existuje predpovedač P pre i-ty bit, všetky bity seedu mimo I_i zafixujeme a zadrátujeme do P. Ukážeme, že zvyšok sa dá dopočítať nie príliš veľkým obvodom, čo bude spor s tvrdením, že f je pekelne ťažká.

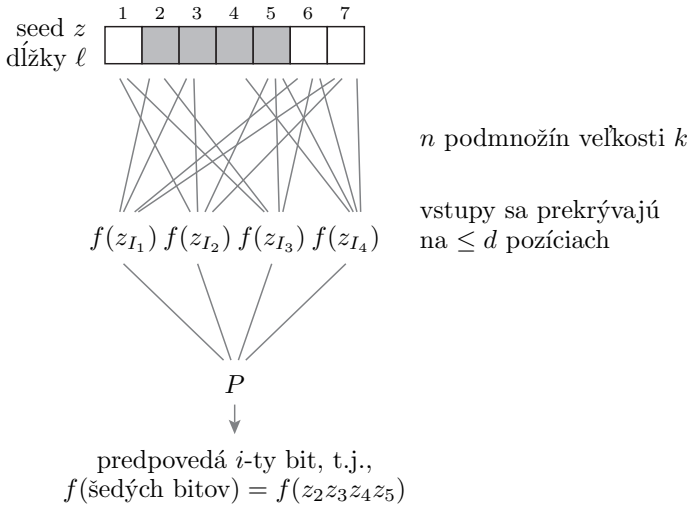
■ **Dôkaz.** Kvôli sporu predpokladajme, že existuje predpovedač P veľkosti $2m^3 = 2^{3d/10+1}$, teda z definície:

$$\Pr_{z \in_R \{0,1\}^\ell}[P(f(z_{I_1}, \ldots, f(z_{I_{i-1}})) = f(z_{I_i})] \geq \frac{1}{2} + \frac{1}{10m}.$$

Označme f_j funkciu, ktorá si vytiahne tie správne bity (z množiny I_j) a na tie aplikuje funkciu f, teda $f_j(z) = f(z_{I_j})$:

$$\Pr_{z \in_R \{0,1\}^\ell}[P(f_1(z), \ldots, f_{i-1}(z)) = f_i(z)] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}}.$$

Otázka znie: pomôže nám pri výpočte f na bitoch z I_i vedieť hodnoty f na vstupoch $I_1, \ldots, I_{i-1}$ (pozri obr. 17.7)?



Obr. 17.7: Predpovedač P, ktorý predpovedá $f(z_{I_i})$ z hodnôt $f(z_{I_j})$ pre $j < i$. (Použili sme generátor s dizajnom z obr. 17.5b.)

Množiny $I_1, \ldots, I_i$ sú všetky rôzne, ale majú neprázdny prienik. Označme teda $x = z_{I_i}$ bity v I_i a z^* tie zvyšné:

$$\Pr_{x \in_R \{0,1\}^k, z^* \in_R \{0,1\}^{\ell-k}} [P(f_1(x,z^*), \ldots, f_{i-1}(x,z^*)) = f(x)] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}}.$$

Opäť, podľa priemerovacieho argumentu existuje konkrétne z^*, pri ktorom je pravdepodobnosť rovnaká alebo väčšia. Tieto bity vieme zadrátovať do funkcií f_j (označme takúto funkciu f_j^*):

$$\Pr_{x \in_R \{0,1\}^k} [P(f_1^*(x), \ldots, f_{i-1}^*(x)) = f(x)] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}}.$$

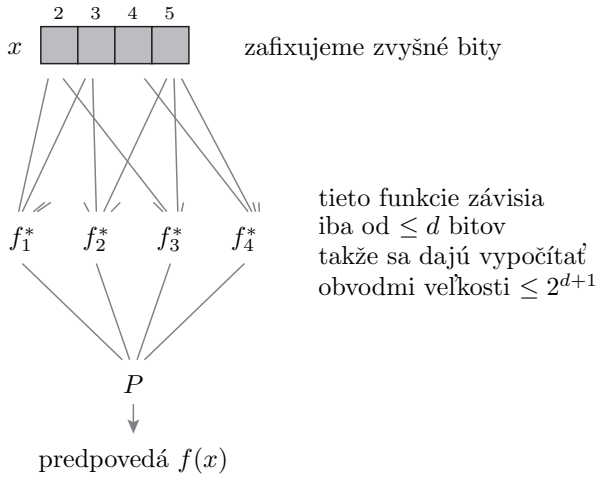
Ostáva uvedomiť si, že každé dve množiny v dizajne majú prienik veľkosti $\leq d$ a teda funkcie f_j^* v skutočnosti závisia iba od $\leq d$ bitov vstupu(!) (pozri obr. 17.8).

Ale každú funkciu d bitov vieme triviálne vypočítať obvodom g_j veľkosti 2^{d+1}. Nahraďme teda funkcie f_j^* obvodmi g_j – dostaneme tak obvod

$$z \mapsto P(g_1(z), \ldots, g_{i-1}(z))$$

veľkosti $m \cdot 2^{d+1} + 2^{3 \cdot d/10+1} = 2^{1.4d+2}$, čo je menej ako $2^{2d} \leq 2^{\gamma k}$. Ak by teda existoval predpovedač, vedeli by sme pomocou neho vytvoriť takýto obvod, ktorý počíta f na viac ako $\frac{1}{2} + \frac{1}{10m} \geq \frac{1}{2} + \frac{1}{S}$ vstupoch. □

Zovšeobecnenie lemy 17.2. *Ak $\mathcal{I}$ je (ℓ, k, d)-dizajn veľkosti $2^{d/10}$ a $f : \{0,1\}^k \to \{0,1\}$ je pekelne ťažká ($\mathsf{H}_{\text{avg}}(f) > 2^{2d} = 2^{\gamma k}$), tak distribúcia $NW_{\mathcal{I}}^f(U_\ell)$ je $\mathsf{H}_{\text{avg}}(f)/10$-pseudonáhodná.*



Obr. 17.8: Zafixujeme ostatné bity seedu a funkcie f_j^* vypočítame triviálnym obvodom veľkosti $O(2^d)$; spolu s predpovedačom P tak dostaneme obvod, ktorý počíta f na dostatočne veľkej časti vstupov, čo je spor s pekelnou ťažkosťou f.

17.4 Sady množín s malými prienikmi

Začnime s „hračkárskym“ príkladom, s ktorým ste sa možno už stretli: v hre *Dobble* (známej tiež pod menom *Spot It!*) máme sadu kariet; na každej je 8 rôznych obrázkov. Čo je však zaujímavé: každá dvojica kariet má práve *jeden* spoločný obrázok (pozri obr. 17.9).

Na obr. 17.9b je vysvetlená konštrukcia takejto sady kariet: ak naše množiny budú priamky (v nejakom konečnom poli $\mathbb{F}_p$), tak každé dve priamky sa pretnú najviac v jednom bode. Množín (priamok) je rádovo p^2 a bodov tiež. Ak chceme viac množín, potrebujeme povoliť väčšie prieniky. Táto konštrukcia sa však dá zovšeobecniť, ak využijeme polynómy malého stupňa: polynómy stupňa d sa pretínajú najviac v d bodoch a je ich rádovo p^{d+1}.

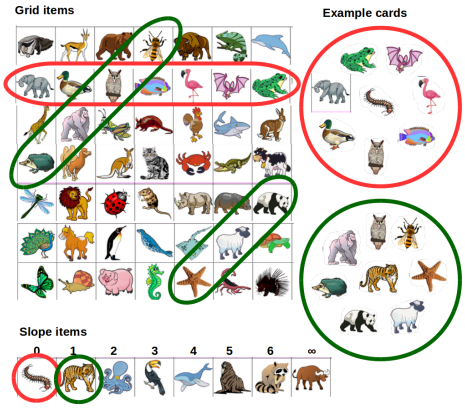
Lema 17.3. *Pre každé $k-1 \geq d \geq 0$ a $\ell \geq 2k^2$ existuje (ℓ, k, d)-dizajn veľkosti aspoň $(\ell/2k)^{d+1}$.*

■ **Dôkaz.** Nech p je prvočíslo medzi $\ell/2k$ a ℓ/k a zvoľme nejakú k-prvkovú podmnožinu $X \subseteq \mathbb{F}_p$. Body budú $X \times \mathbb{F}_p$ a množiny budú grafy polynómov stupňa d, teda množiny tvaru $\{(x, p(x)) \mid x \in X\}$. Polynóm stupňa d má najviac d koreňov a teda dva rôzne polynómy stupňa d sa pretínajú najviac v d bodoch (priesečníky sú korene ich rozdielu). Počet takýchto množín je rovný počtu polynómov, $p^{d+1} > (\ell/2k)^{d+1}$. □

Konkrétne pre

- $\ell = 2k^2 = 2c^2 \log^2 m$,

(a) Hra *Dobble*. Na každej kartičke je 8 druhov obrázkov a každé 2 kartičky majú práve jeden spoločný obrázok. Napríklad na dvoch horných je fialový vták, na dvoch dolných kvet, na dvoch ľavých žltý pes a na dvoch pravých korytnačka. Môžete skúsiť nájsť spoločné obrázky aj na ďalších dvojiciach kariet.



(b) Konštrukcia: Zoraďme si obrázky do tabuľky 7×7 plus 8 špeciálnych obrázkov označených $0, 1, 2, \ldots, 6, \infty$. Každá kartička bude „priamka“ v našej tabuľke (množina tvaru $\{ax + b \mid x \in \mathbb{Z}_7\}$), spolu s obrázkom sklonu priamky (číslo a). Napríklad prvá, červená kartička je horizontálna priamka so sklonom nula; druhá, zelená kartička je šikmá priamka so sklonom 1, špeciálne stĺpce – zvislé priamky budú mať sklon „∞“. Každé dve priamky s rôznym sklonom sa pretínajú práve v jednom bode a dve priamky s rovnakým sklonom majú spoločný práve obrázok pre tento sklon.

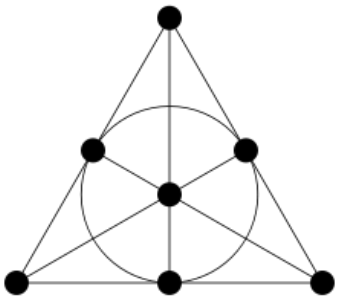
Obr. 17.9: $(57, 8, 1)$-dizajn v hre Dobble. Ide o príklad tzv. projektívnej roviny, kde karty sú „priamky“ a každé dve priamky sa pretínajú práve v jednom bode.

- $k = c \log m$, kde $c = 20/\gamma$ a
- $d = \log m$

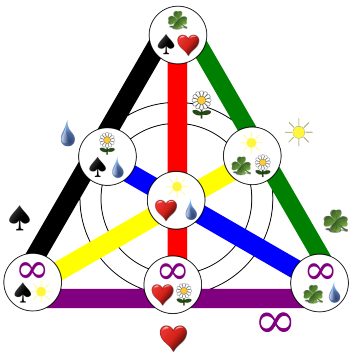
dostávame dizajn veľkosti viac ako $k^d > m$. (V skutočnosti by stačilo $\ell = \Theta(\frac{\log^2 m}{\log \log m})$, čo je však stále veľa.) Toto ešte stále nie sú parametre, s ktorými by sme boli spokojní: Veľkosť seedu až $\log^2 m$ znamená, že všetkých seedov je až $2^{\log^2 m} = m^{\log m}$, čo síce stačí na $\mathsf{BPP} \subseteq \mathsf{QuasiP}$, ale nestačí na úplnú derandomizáciu $\mathsf{BPP} = \mathsf{P}$.

Existujú dizajny s lepšími parametrami? Ukážeme, že áno:

Lema 17.4. *Pre každé $\ell \geq 10k^2/d$ a $k > d$ existuje (ℓ, k, d)-dizajn veľkosti aspoň $m = 2^{d/10}$. Navyše existuje algoritmus, ktorý pre dané (ℓ, k, d) takýto dizajn vygeneruje v čase $2^{O(\ell)}$.*



(a) Fanova rovina – 7 bodov, 7 priamok. Každá priamka má 3 body, každé dve sa pretínajú v jednom bode a duálne cez každé dva body vedie práve jedna priamka.



(b) Mini-verzia hry Dobble so 7 kartami. Tu používame duálny pohľad: karty sú body a priamky obrázky; priamka ide cez body–karty, ktoré obsahujú daný obrázok.

Obr. 17.10: Iný známy príklad je (7,3,1)-dizajn a.k.a. Fanova rovina. Na obr. (b) je mini-verzia hry Dobble, ktorá jej zodpovedá.

Inými slovami, ak chceme dizajn veľkosti m, potrebujeme $d = 10 \log m$; zároveň ak $\ell = c \cdot \log m$, tak $k \leq \sqrt{c} \log m$.

■ **Dôkaz.** Greedy: Začneme s prázdnou rodinou a v každom ďalšom kroku prehľadáme všetky k-prvkové podmnožiny $[\ell]$ a pridáme prvú, ktorej prienik s každou doteraz vybratou množinou má veľkosť najviac d. Toto je zjavne v $2^{O(\ell)}$; ostáva dokázať, že ak vyberieme ľubovoľné množiny $\{I_1, \ldots, I_m\}$ pre $m < 2^{d/10}$, vieme vybrať ďalšiu.

Využijeme pravdepodobnostný argument: predstavme si, že vyberieme prvky I náhodne tak, že každý prvok $x \in [\ell]$ zvolíme nezávisle s pravdepodobnosťou $2k/\ell$. Takto bude $E[|I|] = 2k$ a $E[|I \cap I_j|] = 2k^2/\ell \leq d/5$. Z Černofovej nerovnosti potom vyplýva, že $\Pr[|I| \leq k] \leq 0.1$ (dokonca exponenciálne malá) a $\Pr[|I \cap I_j| \geq d] \leq 1/2^{d/10+1}$. Keďže vybratých množín je $< 2^{d/10}$, pravdepodobnosť, že pre aspoň jednu bude prienik príliš veľký je najviac $2^{d/10}$-krát toľko, čiže $\leq \frac{1}{2}$.

Dokopy je pravdepodobnosť, že I nevyhovuje najviac 0.6 a teda menej ako 1. Inými slovami, je nenulová pravdepodobnosť, že I je dostatočne veľká a má malý prienik s každou doteraz vybratou množinou – a teda taká množina musí existovať. □

Spojením lemy 17.2 a 17.4 dostávame kýžený výsledok:

Veta 17.2 (Nisan a Wigderson (1994)). *Ak existuje pekelne ťažká funkcia v E, tak $\mathsf{BPP} = \mathsf{P}$.*

Všeobecnejšie, ak existuje funkcia $f \in \mathsf{E}$ taká, že $\mathsf{H}_{\text{avg}}(f)(k) \geq S(k)$, tak existuje $S'(\ell)$-pseudonáhodný generátor, kde $S'(\ell)$ je „o čosi menej ako $S(\ell)$“.

Konštrukcie PNG sa postupne zlepšovali a s nimi vzťah medzi ťažkosťou f a predĺžením $S'(\ell)$. Konkrétne pôvodný Nisan-Wigdersonov výsledok bol $S'(\ell) \geq S(k)^\delta$ pre $k \geq \delta\sqrt{\ell \log S(k)}$ po sérii vylepšení Umans (2003) dosiahol $S'(\ell) = S(\delta\ell)^\delta$ pre nejakú konštantu $\delta > 0$.

Veta 17.3 (Umans (2003)). *Ak existuje funkcia $f \in \mathsf{E}$ taká, že že $\mathsf{H}_{\mathrm{avg}}(f)(k) \geq S(k)$ pre každé k, tak existuje $S(\delta\ell)^\delta$-pseudonáhodný generátor pre nejakú konštantu $\delta > 0$.*

Dôsledok 17.2. *Ak existuje $f \in \mathsf{E}$ taká, že*

- $\mathsf{H}_{\mathrm{avg}}(f) = 2^{\Omega(n)}$, *tak* $\mathsf{BPP} = \mathsf{P}$,
- $\mathsf{H}_{\mathrm{avg}}(f) = 2^{n^{\Omega(1)}}$, *tak* $\mathsf{BPP} \subseteq \mathsf{QuasiP}$,
- $\mathsf{H}_{\mathrm{avg}}(f) = n^{\omega(1)}$, *tak* $\mathsf{BPP} \subseteq \mathsf{SUBEXP}$.

Úlohy

- Dokážte, že pre každé dostatočne veľké n existuje funkcia $f : \{0,1\}^n \to \{0,1\}$ s $\mathsf{H}^{0.6}_{\mathrm{avg}}(f) \geq 2^{n/10}$ (t.j. ak ju chceme vypočítať správne aspoň na 60%, potrebujeme obvod veľkosti $2^{n/10}$).

- V tejto kapitole sme ukázali, že ak existujú ťažké funkcie, vieme z nich vytvoriť dobrý pseudonáhodný generátor. Ukážte, že to platí aj naopak – ak existuje dobrý PNG, tak E obsahuje ťažké funkcie. Presnejšie, dokážte, že ak existuje $S(\ell)$-PNG, tak existuje $f \in \mathsf{E}$ taká, že $\mathsf{H}_{\mathrm{wrs}}(f)(n) \geq S(n)$. (Hint: Pre PNG G so seedom dĺžky ℓ vezmime len prvých $\ell + 1$ bitov; zistiť, či takýto reťazec G vygeneruje je ťažké; formálne: pre $|x| = \ell + 1$ nech $f(x) = 1$, ak $\exists z \in \{0,1\}^\ell : x$ je prefix $G(z)$.)

Literatúra

Doron, Dean, Dana Moshkovitz, Justin Oh, a David Zuckerman. 2019. "Nearly optimal pseudorandomness from hardness." Tech. zpr., ECCC preprint TR19-099.

L'Ecuyer, Pierre a Richard Simard. 2007. "TestU01: A C library for empirical testing of random number generators." *ACM Transactions on Mathematical Software (TOMS)* 33(4), s. 1–40.

Marsaglia, George. 1996. "DIEHARD: a battery of tests of randomness." .

Nisan, Noam a Avi Wigderson. 1994. "Hardness vs randomness." *Journal of computer and System Sciences* 49(2), s. 149–167.

Rukhin, Andrew, Juan Soto, James Nechvatal, Miles Smid, a Elaine Barker. 2001. "A statistical test suite for random and pseudorandom number generators for cryptographic applications." Tech. zpr., Booz-allen and hamilton inc mclean va.

Umans, Christopher. 2003. "Pseudo-random generators for all hardnesses." *Journal of Computer and System Sciences* 67(2), s. 419–440.

Kapitola 18

Veľmi ťažké funkcie z ťažkých

V predchádzajúcej kapitole sme si ukázali, ako z *veľmi ťažkých* (potrebujeme exponenciálny obvod, ak ju chceme spočítať na 99%) dokážeme vyrobiť *pekelne ťažké* funkcie (potrebujeme exponenciálny obvod, ak ju chceme spočítať čo i len na $\frac{1}{2} + \frac{1}{S}$ vstupoch). V tejto kapitole si ukážeme, ako vyrobiť veľmi ťažké funkcie z ťažkých v najhoršom prípade, teda takých, ktoré je ťažké spočítať úplne presne, na 100%.

Tým zakončíme našu dlhú púť, ktorá dokazuje, že ak existujú ťažké funkcie, potom existujú veľmi ťažké aj pekelne ťažké funkcie, z nich vieme spraviť dobrý pseudonáhodný generátor a derandomizovať BPP.

Veta 18.1 (Veľmi ťažké funkcie z ťažkých v najhoršom prípade). *Ak* E *obsahuje ťažkú funkciu v najhoršom prípade, tak obsahuje aj veľmi ťažkú funkciu.*

Presnejšie, ak $f \in \mathsf{E}$ *a* $\mathsf{H}_{\mathrm{wrs}}(f)(n) \geq S(n)$*, tak existuje* $\hat{f} \in \mathsf{E}$ *a* $c > 0$ *také, že* $\mathsf{H}^{0.99}_{\mathrm{avg}}(\hat{f})(n) \geq S(n/c)/n^c$ *pre všetky (až na konečne veľa)* n*. Všimnite si, že ak* $S(n)$ *je exponenciálna alebo superpolynomiálna, tak aj* $S(n/c)/n^c$ *rastie exponenciálne, respektíve superpolynomiálne.*

■ **Dôkaz.** Máme funkciu f a chceme $\hat{f}$ takú, že

- ak f je ťažká v najhoršom prípade,
- tak $\hat{f}$ je veľmi ťažká.

Inými slovami:

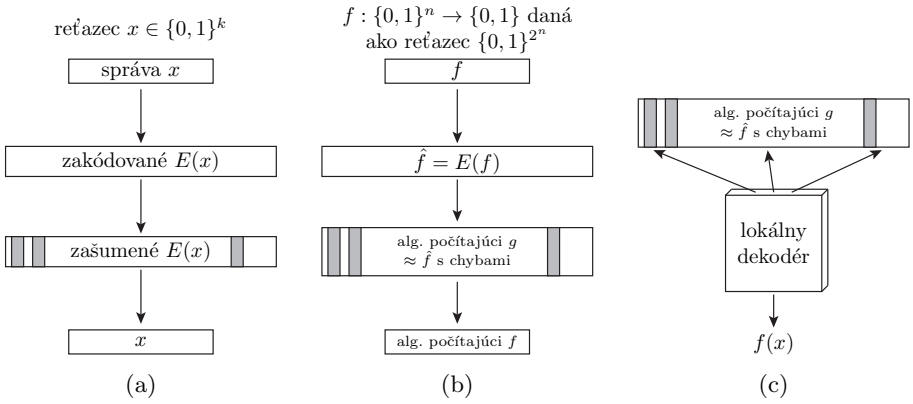
- keby sa $\hat{f}$ dala vypočítať na 99% vstupov
- tak by sa f dala spočítať všade (nie o moc väčším obvodom)

Nepripomína vám to niečo?

...

Samoopravné kódy (pozri obrázok 18.1)! Pri samoopravných kódoch chceme x zakódovať na reťazec $\hat{x}$ tak, že

- ak nám niekto pošle $\hat{x}$ a pri prenose možno vzniklo najviac 1% chýb,
- tak vieme zrekonštruovať pôvodné x úplne presne.



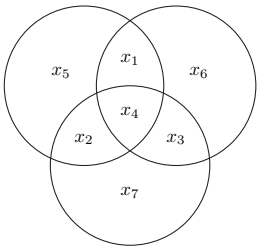
Obr. 18.1: (a) Samoopravné kódy (SOK) sú spôsob, akým vieme zakódovať správu x tak, aby sme ju vedeli rekonštruovať aj v prípade, že sa pri prenose vyskytnú chyby. (b) Funkciu $f : \{0,1\}^n \to \{0,1\}$ môžeme reprezentovať jej tabuľkou hodnôt, čo je reťazec dĺžky 2^n. Tento reťazec zakódujeme SOK a výsledok interpretujeme ako funkciu $\hat{f}$ (na väčšom vstupe). Vďaka samoopravnej vlastnosti kódov vieme rekonštruovať hodnoty f, aj keď máme iba algoritmus, ktorý počíta $\hat{f}$ približne, s chybami. (c) Na rozdiel od tradičných SOK, kde sa dekóduje celá správa x naraz, budeme potrebovať tzv. lokálne dekódovateľné kódy, kde vieme efektívne dekódovať jednotlivé bity správy. Lokálny dekodér je algoritmus, ktorý dostane g ako čiernu krabičku; funkciu g vyhodnotí v niekoľkých bodoch $g(a_1), \ldots, g(a_m)$ (pre väčšinu bodov bude $g(a_i) = \hat{f}(a_i)$, ale niektoré hodnoty môžu byť chybné) a z nich odvodí výsledok $f(x)$.

Uvažujme reťazec $F \in \{0,1\}^N$ dĺžky $N = 2^n$ všetkých hodnôt funkcie $f(x)$ pre $x \in \{0,1\}^n$. Zakódujme ho samoopravným kódom $E : \{0,1\}^N \to \{0,1\}^{N^C}$. Reťazec $\hat{F} = E(F)$ zase interpretujeme ako funkciu $\hat{f} : \{0,1\}^{C \cdot n} \to \{0,1\}$. Tvrdíme, že ak f bola funkcia ťažká v najhoršom prípade, potom $\hat{f}$ je veľmi ťažká, pretože ak by sme vedeli počítať $\hat{f}$ s $\leq 1\%$ chýb, tak by sme pomocou dekódovacieho algoritmu dokázali chyby opraviť a počítať funkciu f úplne presne.

Jediný podstatný rozdiel pri dekódovaní je ten, že „klasické“ dekódovacie algoritmy načítajú celý zašumený reťazec $\hat{x}$ a opravia v ňom chyby. My však chceme dokázať, že ak existuje nejaká funkcia g, ktorá má malý obvod a počíta $\hat{f}$

0	1	1	0	**1**	0	1	0
1	**0**	**1**	**1**	**1**	**1**	**0**	**0**
1	0	1	1	**0**	1	1	1
1	1	0	0	**1**	1	0	0
1	1	1	0	**1**	1	1	0
0	0	1	0	**1**	0	1	1
1	1	0	1	**0**	1	0	0
1	0	1	1	**0**	1	0	0

(a) Obdĺžnikový $(8^2, 7^2)$-kód: bity v poslednom riadku a stĺpci sme doplnili tak, aby bol v každom riadku aj stĺpci párny počet jednotiek. Takto vieme ľahko nájsť a opraviť jednu chybu – v príklade na obr. je chybný bit v 2. riadku a 5. stĺpci (nesedí parita).



$x_1 \oplus x_2 \oplus x_4 \oplus x_5 \equiv 0$

$x_1 \oplus x_3 \oplus x_4 \oplus x_6 \equiv 0$

$x_2 \oplus x_3 \oplus x_4 \oplus x_7 \equiv 0$

(b) Hammingov (7,4)-kód: K pôvodnej správe $x_1x_2x_3x_4$ pridáme 3 bity tak, aby bol v každom kruhu párny súčet. Takýto kód dokáže opraviť jednu chybu. Napríklad, ak parita nesedí v horných dvoch kruhoch, ale v dolnom sedí, musí byť chybný x_1.

Obr. 18.2: Dva jednoduché kódy, ktoré dokážu opraviť jednu chybu.

na 99%, tak aj f má nie o moc väčší obvod. A na to si nemôžeme dovoliť načítať celý reťazec G, ktorý je exponenciálne dlhý a na ňom ešte spúšťať polynomiálne algoritmy. Ideálne by bolo, keby sme každý jeden bit x vedeli zrekonštruovať iba z malého počtu bitov $\hat{x}$. Takýto dekodér voláme lokálny (pozri obrázok 18.1c).

Keď si to teda zhrnieme, potrebujeme efektívny samoopravný kód s lokálnym dekodérom, ktorý

- vie opraviť 1% chýb,
- kódovanie je v čase $\mathsf{poly}(N)$,
- dekódovanie je v čase $\mathsf{polylog}(N) = \mathsf{poly}(n)$.

Takýto kód postupne zostrojíme vo zvyšku kapitoly.

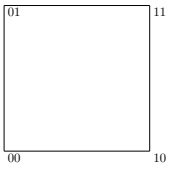
Ak $f \in \mathsf{E}$, potom g je tiež v E ($2^n \times$výpočet f trvá $2^n \cdot 2^{O(n)} = 2^{O(n)}$ a kódovanie $\mathsf{poly}(2^n) = (2^n)^k = 2^{O(n)}$). Keby existoval obvod veľkosti $s(n)$ počítajúci g na 99% vstupov, tak g + lokálny dekodér počíta f presne a má veľkosť $\mathsf{poly}(n) \cdot s(n)$. □

18.1 Samoopravné kódy

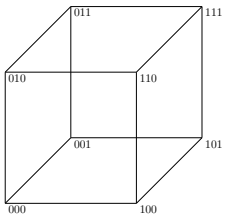
Samoopravný kód je zobrazenie $E : \{0,1\}^k \to \{0,1\}^n$ (alebo všeobecnejšie $E : \Sigma^k \to \Gamma^n$) také, že ak dostaneme zašumenú správu y, ktorá je blízko nejakého kódu $E(x)$, tak pôvodnú správu x vieme nájsť. Na to stačí, aby sa pre každé dve správy $x \neq x'$ ich kódy $E(x), E(x')$ líšili na veľa pozíciach.

Definícia 18.1. *Hammingova vzdialenosť dvoch rovnako dlhých reťazcov $x, y \in \Sigma^n$ je počet pozícií, na ktorých sa líšia:*

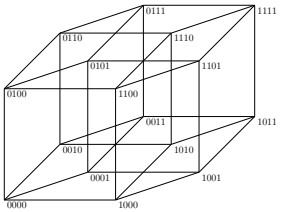
$$\Delta(x, y) = |\{i \mid x_i \neq y_i\}|.$$



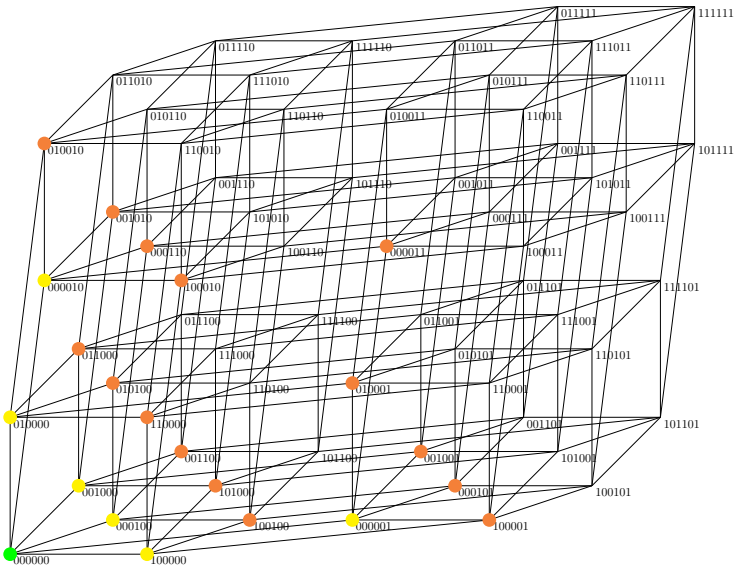
(a) $\{0,1\}^2$ je štvorec, čiže 2D kocka. Ešte jednoduchší prípad je 1D kocka, čo sú len dva body spojené hranou: 0—1. Štvorec získame z dvoch kópií 0—1, v jednej kópii pridáme na koniec 0, v druhej 1 a prislúchajúce vrcholy spojíme.



(b) $\{0,1\}^3$ je kocka. Predná stena sú reťazce končiace nulou, zadná stena sú reťazce končiace jednotkou. Kocku získame z dvoch kópií 2D kocky pospájaním prislúchajúcich vrcholov.



(c) $\{0,1\}^4$ je 4D hyperkocka. Jej steny tvoria 3D kocky: Všimnite si prednú stenu, ktorú tvoria všetky 3-bitové reťazce a na konci nula a zadnú stenu, ktorú tvoria všetky 3-bitové reťazce a na konci jednotka.



(d) $\{0,1\}^6$ je 6D hyperkocka. Na ľavej strane vidíme reťazce končiace nulou, vpravo reťazce končiace jednotkou, respektíve, vidíme, ako sa 6D kocka skladá zo štyroch 4D kociek: vľavo dolu sú reťazce končiace 00, vpravo dolu končiace 01, vľavo hore 10 a vpravo hore 11. Na obrázku je tiež vyznačený bod 000000 vľavo dolu a všetky body vo vzdialenosti ≤ 2 od neho, takejto množine väčšinou hovoríme *guľa* so stredom v 000000 a polomerom 2.

Obr. 18.3: Metrický priestor $\{0,1\}^n$ si môžeme predstaviť tak, že pospájame vrcholy líšiace sa v jednom bite. Dostaneme tak n-rozmernú hyperkocku, pričom Hammingova vzdialenosť dvoch reťazcov sa rovná vzdialenosti (počtu hrán) medzi dvoma vrcholmi.

Relatívna Hammingova vzdialenosť je $\delta(x,y) = \Delta(x,y)/n$.

Definícia 18.2. *Hovoríme, že* $E : \Sigma^n \to \Sigma^m$ *je samoopravný kód (SOK) so vzdialenosťou* $d \in [0,1]$ *nad abecedou* Σ*, ak* $\forall x \neq y : \delta(E(x), E(y)) \geq d$.

Na obrázku 18.2 sú dva príklady veľmi jednoduchých kódov, ktoré dokážu opravovať jednu chybu. My budeme potrebovať opravovať oveľa viac. Postupne si ukážeme tri lepšie samoopravné kódy. Výsledný kód dostaneme ich kombináciou.

Hadamardov kód

Variant Hadamardovho kódu používala napríklad americká sonda Mariner 9 vyslaná na obežnú dráhu Marsu. Fotografie Marsu v kvalite 6 bitov na pixel (64 odtieňov šedej) kódovala 32 bitmi na pixel a posielala ich na Zem. Vďaka tomuto kódu sa dalo opraviť až 7 chybných bitov v každom pixeli.

Definícia 18.3 (Hadamardov kód). *Hadamardov kód* $\mathsf{H} : \{0,1\}^n \to \{0,1\}^{2^n}$ *je SOK definovaný* $\mathsf{H}(x)_i = \langle x, i\rangle = \sum_j x_j i_j$.

Napríklad vektor `101` sa zakóduje na `01011010`.

pozícia	hodnota
000	0
001	1
010	0
011	1
100	1
101	0
110	1
111	0

Veta 18.2. *Hadamardov kód má vzdialenosť* $1/2$ *a existuje lokálny dekodér opravujúci* $< 1/4$ *chýb v čase* $O(n)$.

■ **Dôkaz.** Všimnite si, že Hadamardov kód je lineárny ($\mathsf{H}(x \oplus y) = \mathsf{H}(x) \oplus \mathsf{H}(y)$) a teda rozdiel

$$\min_{x,y} \delta(\mathsf{H}(x), \mathsf{H}(y)) = \min_{x,y} \delta(\mathsf{H}(x-y), \mathsf{H}(0)) = \min_{z} \delta(\mathsf{H}(z), 0)$$

vždy vieme posunúť do nuly a vzdialenosť dvoch najbližších vektorov je taká istá ako najmenšia váha nenulového vektoru. Stačí teda dokázať, že kód každého nenulového vektoru má polovicu jednotiek, čo je zrejmé, viď príklad vyššie: vektor `101` má prvý bit jednotku a teda pozície začínajúce sa na `1` budú mať opačnú hodnotu ako tie začínajúce sa na `0`, takže núl a jednotiek bude rovnako.

Predstavme si teraz, že máme `01011010` a chceme dekódovať prvý bit vstupu. Môžeme použiť rovnaký arguement ako vyššie: ak sa pozrieme na pozície `0**` vs. zodpovedajúce pozície `1**`, vidíme, že hodnota na výstupe sa líši, takže prvý

bit musel byť `1`. Naopak ak sa pozrieme na pozície `*0*` a zodpovedajúce pozície `*1*`, zistíme, že zmena iba v druhom bite nemá vplyv na výstup, preto druhý bit musel byť `0`.

Vo všeobecnosti, ak chceme dekódovať j-ty bit, nech e^j je vektor, ktorý má jednotku iba na j-tej pozícii, zvyšok sú nuly. Zvolíme si náhodný vektor y, pozrieme sa na pozíciu y a $y \oplus e^j$ a tieto bity zoxorujeme. Ak f je nejaký reťazec dĺžky 2^n (predstavme si ho ako funkciu $f : \{0,1\}^n \to \{0,1\}$) a predpokladajme, že existuje x také, že vzdialenosť f a $\mathsf{H}(x)$ je najviac $\rho < 1/4$, tzn. $\Pr_y[f(y) \neq \langle x, y\rangle] \leq \rho$. Potom hodnota $f(y)$ je zle s pravdepodobnosťou ρ, $f(y + e^j)$ je zle s pravdepodobnosťou ρ, takže obe sú správne s pravdepodobnosťou $1 - 2\rho$ a vtedy $f(y) \oplus f(y \oplus e^j) = (\langle x, y\rangle) \oplus (\langle x, (y + e^j)\rangle) = 2(\langle x, y\rangle) \oplus (\langle x, e^j\rangle) = \langle x, e^j\rangle$. Teda s pravdepodobnosťou $1 - 2\rho \geq 1/2$ dáme správny výsledok. Pre väčšiu úspešnosť môžeme zvoliť viacero y a zvoliť väčšinovú odpoveď. □

Vzdialenosť je super. Ešte raz: kódy každých dvoch rôznych správ sa líšia až v polovici bitov. Jediná nevýhoda je, že vstup sa až príliš nafúkne.

Reed-Solomonov kód

Reed-Solomonov kód je veľmi praktický kód, ktorý sa v využíva pri kódovaní údajov na CDčkach, DVDčkach, Blu-ray diskoch, či pri ukladaní na disk (schéma RAID 6), kóduje sa ním obsah QR kódov, atď. Tento kód používali aj vesmírne misie ako Voyager, Galileo, Cassini, Mars Pathfinder a iné.

RS kód využíva väčšiu abecedu. Jednotlivé symboly interpretujeme ako prvky poľa $\mathbb{F}_q$ (pre jednoduchosť si môžete predstaviť, že q je prvočíslo, potom $\mathbb{F}_q = (\mathbb{Z}_q, +, \cdot)$, teda celé čísla mod q). Základná myšlienka je, že pôvodnú správu dĺžky d interpretujeme ako d bodov, ktoré jednoznačne určujú nejaký polynóm p stupňa $d - 1$ (pozri obrázok 18.4). Správu zakódujeme tak, že k nej pridáme ďalšie body, ktoré ležia na tomto polynóme. Ukážeme si, že ak máme napríklad $2d + 1$ bodov a z toho menej ako $1/4$ je chybných, stále dokážeme polynóm stupňa d zrekonštruovať.

Definícia 18.4 (Reed-Solomonov kód). *Nech $\mathbb{F}_q$ je pole, $d \leq n \leq q$ a nech $a_0, \ldots, a_{n-1}$ je n prvkov poľa $\mathbb{F}_q$. Správu $c_0 \ldots c_{d-1}$ interpretujme ako koeficienty polynómomu*

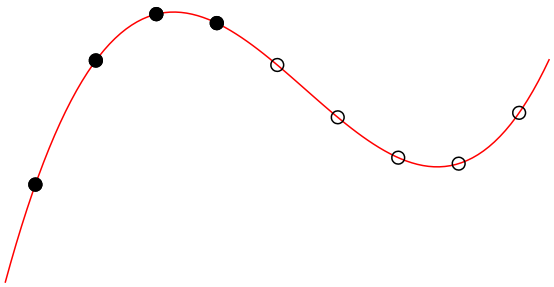
$$p(x) = \sum_i c_i x^i.$$

Reed-Solomonov kód $\mathsf{RS} : \mathbb{F}_q^d \to \mathbb{F}_q^n$ je zobrazenie

$$c_0 \ldots c_{d-1} \mapsto p(a_0) \ldots p(a_{n-1}).$$

Reed-Solomonov kód je tiež lineárny, má vzdialenosť $1 - d/n$ (dva polynómy d-teho stupňa, ktoré sa zhodujú v $d + 1$ bodoch sa zhodujú).

Veta 18.3 (Gemmell a Sudan (1992)). *Daný je zoznam dvojíc $(a_1, b_1), \ldots, (a_n, b_n)$ z $\mathbb{F}_q^2$, pričom existuje polynóm $p : \mathbb{F}_q \to \mathbb{F}_q$ stupňa d, pre ktorý $p(a_i) = b_i$ pre $t > n/2 + d/2$ dvojíc. Existuje algoritmus, ktorý dokáže p zrekonštruovať v polynomiálnom čase.*



Obr. 18.4: Reed-Solomonov $(9, 4)$-kód. Štyri body (alebo štyri koeficienty) jednoznačne špecifikujú kubický polynóm. Zakódovaná správa bude pozostávať z 9-tich bodov, ktoré ležia na tomto polynóme. Takýto kód dokáže opraviť 3 chyby. (Na obrázku je pre ilustráciu polynóm nad $\mathbb{R}$ – polynómy nad konečnými poľami $\mathbb{F}_q$ vyzerajú, samozrejme, trochu inak, ale fungujú rovnako.)

■ **Dôkaz.** Rovnica $p(a_i) = b_i$ pre $k = n - t = \lfloor n/2 - d/2 \rfloor$ bodov nie je splnená, ale nevieme, pre ktoré. Predstavme si, že máme polynóm e, tzv. chybový polynóm, v ktorom sú zakódované tieto miesta tak, že ak $p(a_i) \neq b_i$, tak $e(a_i) = 0$. Ak všetky rovnice prenásobíme $e(a_i)$, dostaneme

$$e(a_i)p(a_i) = e(a_i)b_i, \qquad (*)$$

tieto rovnice však už platia *pre všetky* i – tam, kde $e(a_i) = 0$, sa $p(a_i)$ a b_i môžu líšiť a aj tak dostaneme $0 = 0$, ale vo zvyšných bodoch, kde $e(a_i) \neq 0$, môžeme $e(a_i)$ z oboch strán vykrátiť a platí $p(a_i) = b_i$.

Úlohu teda môžeme preformulovať nasledovne: hľadáme polynóm p stupňa d a nenulový polynóm e stupňa najviac k také, že $e(a_i)p(a_i) = e(a_i)b_i$ pre všetky i. Nenulový polynóm stupňa k má najviac k núl a teda $p(a_i) = b_i$ pre aspoň t dvojíc.

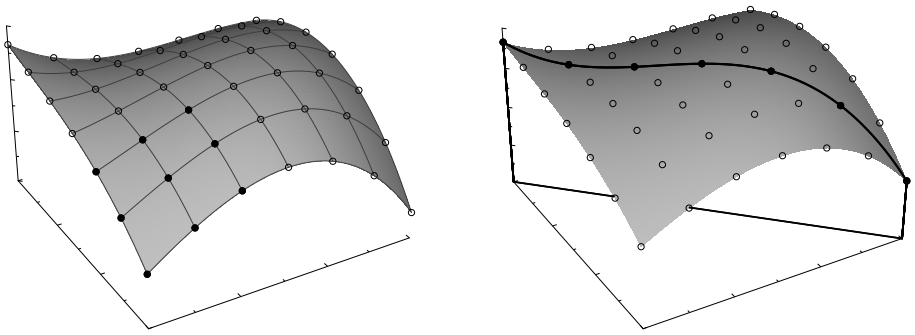
Dostávame tak n rovníc, kde a_i, b_i poznáme a koeficienty e a p sú neznáme. Problém je, že tieto rovnice nie sú lineárne. Avšak súčin $e \cdot p = q$ je opäť polynóm (stupňa $k + d = \lfloor n/2 + d/2 \rfloor$) a namiesto $(*)$ môžeme hľadať riešenie rovníc

$$q(a_i) = e(a_i)b_i.$$

Polynóm p potom získame ako q/e. Keďže e je nenulový polynóm, môžeme zafixovať jeho vedúci koeficient e_k rovný jednej. Ak označíme neznáme koeficienty chybového polynómu $e_0, \ldots, e_{k-1}$ a $q_0, \ldots, q_{k+d}$ koeficienty súčinu $q = e \cdot p$, po úprave dostaneme

$$q_0 + q_1 a_i + q_2 a_i^2 + \cdots + q_{k+d} a_i^{k+d} = e_0 b_i + e_1 a_i b_i + e_2 a_i^2 b_i + \cdots + e_{k-1} a_i^{k-1} b_i + a_i^k b_i,$$

čo je n lineárnych rovníc o $\leq n$ neznámych. (Ak e, q aj e', q' sú riešenia, potom $e/q = e'/q'$: totiž ak $q(a_i) = e(a_i)b_i$ a $q'(a_i) = e'(a_i)b_i$, tak $q(a_i)e'(a_i)b_i = q'(a_i)e(a_i)b_i$ a po vykrátení b_i (platí aj pre $b_i = 0$, pretože vtedy $q(a_i) = q'(a_i) = 0$) $q(a_i)e'(a_i) = q'(a_i)e(a_i)$. Avšak qe' aj $q'e$ sú polynómy stupňa $2k + d$, ktoré sa zhodujú v $n > 2k + d$ bodoch, takže $qe' = q'e$ a teda $q/e = q'/e'$.)



(a) V dvoch rozmeroch 10 bodov (koeficientov) na vstupe špecifikuje kubický polynóm dvoch premenných $p(x,y) = \sum_{i+j\leq 3} c_{ij}x^i y^j$. Zakódovaná správa pozostáva z viacerých bodov, ktoré ležia na $p(x,y)$.

(b) Pri dekódovaní bodu si vyberieme náhodnú priamku, ktorá cez neho prechádza. Ak zúžime funkciu na danú priamku, dostaneme polynóm len *jednej* premennej a úloha sa nám redukuje na dekódovanie RS kódu.

Obr. 18.5: Reed-Mullerov kód s dvoma premennými ($m = 2$). (Na obrázku je pre ilustráciu polynóm nad $\mathbb{R}$ – polynómy nad konečnými poľami $\mathbb{F}_q$ vyzerajú, samozrejme, trochu inak, ale fungujú rovnako.)

□

Nevýhoda pre nás je, že nemá lokálny dekodér.
Ten má až zovšeobecnenie kódu na polynómy vyššieho stupňa:

Reed-Mullerov kód

Definícia 18.5 (Reed-Mullerov kód). *Nech $\mathbb{F}_q$ je pole, $d < q$. Reed-Mullerov kód s parametrami q, m, d je funkcia $\mathsf{RM} : \mathbb{F}_q^{\binom{m+d}{d}} \to \mathbb{F}_q^{q^m}$, ktorá správu c interpretuje ako koeficienty polynómu celkového stupňa d s m premennými nad $\mathbb{F}_q$ a vyhodnotí ho vo všetkých bodoch $\mathbb{F}_q^m$. Tzn. na vstupe dĺžky $\binom{m+d}{d}$ sú koeficienty $\{c_{i_1,\ldots,i_m}\}$, ktoré interpretujeme ako polynóm*

$$P(x_1,\ldots,x_m) = \sum_{i_1+\cdots+i_m\leq d} c_{i_1,\ldots,i_m} x_1^{i_1} x_2^{i_2} \cdots x_m^{i_m}$$

a na výstup dáme postupnosť $\{P(x_1,\ldots,x_m)\}$ pre všetky $x_1,\ldots,x_m \in \mathbb{F}_q$.

Pre $m = 1$ dostávame obyčajný Reed-Solomonov kód; naopak pre $d = 1$ a $q = 2$ dostávame Hadamardov kód.

Veta 18.4. *Reed-Mullerov kód má vzdialenosť $1-d/q$. Lokálne vieme dekódovať správy, ktoré majú menej ako $\rho \leq (1-d/q)/6$ chýb.*

■ **Dôkaz.** Vzdialenosť $1 - d/q$ vyplýva zo Schwartz-Zippelovovej lemy.

Dekódovanie: Na reťazec sa budeme pozerať ako na funkciu $f : \mathbb{F}_q^m \to \mathbb{F}_q$, pričom predpokladáme, že f je blízko nejakého polynómu p stupňa d:

$$\Pr_x[f(x) \neq p(x)] < \rho.$$

Môžeme si to predstaviť tak, že f počíta p, ale s chybami a my by sme chceli algoritmus, ktorý vyhodnotí f v niekoľkých bodoch a na základe nich spočíta $p(x)$, pričom opraví chyby.

Nech $a, b \in \mathbb{F}_q^m$ sú dva body (začiatok a smer), potom definujeme priamku $\ell : \mathbb{F}_q \to \mathbb{F}_q^m$ danú predpisom $\ell_{a,b}(t) = a + tb$. Pre $t_1 \neq t_2 \in \mathbb{F}_q$ sú body $\ell_{a,b}(t_1)$ a $\ell_{a,b}(t_2)$ distribuované nezávisle rovnomerne na $\mathbb{F}_q^m$. Pre každé a a pre každé $t \in \mathbb{F}_q - \{0\}$ ak b vyberieme náhodne, tak $\ell_{a,b}(t)$ je náhodný bod z $\mathbb{F}_q$. Označme $f|_\ell : \mathbb{F}_q \to \mathbb{F}_q$ zúženie funkcie f na priamku ℓ dané predpisom $f|_\ell(t) = f(\ell(t))$. Ak p je polynóm viacerých premenných stupňa d a ℓ je priamka, tak $p|_\ell$ je polynóm jednej premennej stupňa najviac d.

Nech $L = \ell_{x,z}$ je náhodná priamka idúca cez x. Pozrieme si hodnoty $f(x+tz)$ vo všetkých bodoch L. Pomocou dekodéru pre Reed-Solomonov kód zistíme polynóm $r : \mathbb{F}_q \to \mathbb{F}_q$, ktorý sa zhoduje s $f|_L$ na najväčšom počte bodov a odpovieme $r(0)$.

Keďže body $\ell_{x,z}$ pre náhodné z sú distribuované rovnomerne po $\mathbb{F}_q^m$, očakávaný počet bodov na L, kde sa f a p líšia je najviac $\rho \cdot q$. Z Markovovej nerovnosti vyplýva, že s pravdepodobnosťou $2/3$ chýb na L nebude viac ako $3\times$ toľko. Keďže $3 \times \rho q < (1 - d/q)q/2 = q/2 - d/2$, v tom prípade vieme r správne dekódovať: $r = p|_L$ a $r(0) = p(x)$. □

Reed-Mullerov kód je super, jediný problém je, že (podobne ako RS) používa veľkú abecedu. V praxi, napríklad pri Reed-Solomonovom kóde môžeme zvoliť $q = 256$ a prvky kódovať ako 1 byte. Napríklad pre $n = 230$ a $m = 256$ dostávame RS kód opravujúci 13 chýb (5% bytov / 0.6% bitov).

Vo všeobecnosti môžeme 1 prvok $\mathbb{F}_q$ zakódovať pomocou $\log q$ bitov. Problém je, že takýmto spôsobom dostaneme dosť malú vzdialenosť (v prepočte na bity). Môže sa totiž stať, že jeden chybný znak z $\mathbb{F}_q$ bude zodpovedať len jedinému chybnému bitu z $\log q$. Takýmto spôsobom preto nedosiahneme vzdialenosť väčšiu ako $1/\log q$.

Riešenie? Jednotlivé znaky $\mathbb{F}_q$ nebudeme kódovať priamo pomocou $\log q$ bitov, ale opäť použijeme samoopravný kód! Tým pádom zmena v jednom znaku bude zodpovedať veľa chýbam v bitovej sekvencii. Inými slovami: ak kódujeme znaky samoopravným kódom, muselo by nastať veľa chýb (veľa zlých bitov) v jednom znaku, aby sme ho nevedeli dekódovať.

Zložené kódy

Ak máme dva samoopravné kódy (s vhodnými abecedami), môžeme ich zložiť tak, že správu najskôr zakódujeme pomocou E_1 a každý znak výsledku (zo Σ)

zakódujeme pomocou E_2:

$$\begin{aligned} E_1 : &\quad \{0,1\}^n \xrightarrow{E_1} \Sigma^m \\ E_2 : &\quad \Sigma \xrightarrow{E_2} \{0,1\}^k \\ E_2 \circ E_1 : &\quad \{0,1\}^n \xrightarrow{E_1} \Sigma^m \xrightarrow{E_2} \{0,1\}^{mk} \end{aligned}$$

$$E_2 \circ E_1 : x \mapsto E_2(y_1) \cdots E_2(y_m), \qquad \text{kde } y = E_1(x)$$

Aká je vzdialenosť takéhoto kódu? Ak δ_1, δ_2 sú vzdialenosti E_1, E_2, znamená to, že každé dve kódové slová E_1 sa líšia aspoň v δ_1-tine znakov. Tieto odlišné znaky sú ďalej zakódované E_2 a teda každý sa líši aspoň v δ_2-tine znakov. Každé dve slová zakódované $E_2 \circ E_1$ sa tým pádom líšia na aspoň $\delta_1 \cdot \delta_2$-tine pozícií.

Kód, ktorý hľadáme, je $\mathsf{H} \circ \mathsf{RM}$ (prvky poľa $\mathbb{F}_q$ reprezentujeme binárne ako $\{0,1\}^{\log q}$):

$$\begin{aligned} \mathsf{RM} : &\quad \{0,1\}^k \xrightarrow{\mathsf{RM}} \mathbb{F}_q^{q^m} \sim \{0,1\}^{\log q \cdot q^m} \\ \mathsf{H} : &\quad \{0,1\}^{\log q} \xrightarrow{\mathsf{H}} \{0,1\}^q \\ \mathsf{H} \circ \mathsf{RM} : &\quad \{0,1\}^k \xrightarrow{\mathsf{RM}} \{0,1\}^{\log q \cdot q^m} \xrightarrow{\mathsf{H}} \{0,1\}^{q^{m+1}} \end{aligned}$$

Použijeme RM-kód s parametrami

- $q = \log^5 N$
- $m = \log N / \log\log N$
- $d = \log^2 N$

a H-kód s parametrom $n = \log q$, t.j. $\mathsf{H} : \{0,1\}^{\log q} \to \{0,1\}^q$, čo je $\mathsf{H} : \{0,1\}^{5 \log\log N} \to \{0,1\}^{\log^5 N}$.

Všimnite si, že

$$(\log^c N)^{\log N / \log\log N} = \left(2^{c \cdot \log\log N}\right)^{\log N / \log\log N} = 2^{c \cdot \log N} = N^c,$$

takže dĺžka správy bude $\binom{d+m}{m} \geq (d/m)^m > (\log N)^m = N$ a dĺžka kódu $q^{m+1} = O(N^{5+\varepsilon})$.

Zopakujme si, aký kód sme hľadali:

- vie opraviť 1% chýb,
- kódovanie je v čase $\mathsf{poly}(N)$,
- dekódovanie je v čase $\mathsf{polylog}(N) = \mathsf{poly}(n)$.

A čo sme získali?

- H má vzdialenosť $1/2$, pričom lokálne vieme dekódovať $1/4$, RM s týmito parametrami má vdialenosť $1 - \log^2 N / \log^5 N = 1 - 1/n^3$, pričom lokálne vieme dekódovať asi $1/6$ chýb; teda $\mathsf{H} \circ \mathsf{RM}$ vie lokálne opravovať asi $(1/4) \cdot (1/6) = 1/24 - O(1/n^3)$, teda asi 4% chýb,

	H	RS	RM
dĺžka správy	n	d	$\binom{d+m}{m} \geq (d/m)^m \geq N$
dĺžka kódu	2^n	n	$q^m = N^5$
veľkosť abecedy	2	$q \geq n$	$q = \mathsf{polylog}(N)$
vzdialenosť	$1/2$	$1 - d/n$	$1 - d/q \approx 1 - \varepsilon$
koľko chýb vieme lokálne opraviť	$1/4$	—	$1/6 - d/6q \approx 1/6 - \varepsilon$

Tabuľka 18.1: Prehľad parametrov spomenutých kódov. V Reed-Mullerovom kóde sme dosadili parametre vhodné pre našu konštrukciu.

- kódovanie je v polynomiálnom čase $\mathsf{poly}(N)$,
- dekódovanie H je v $O(n) = O(\log q) = \mathsf{polylog}(N)$, dekódovanie RM je v čase $\mathsf{poly}(q, m, d) = \mathsf{polylog}(N)$.

Literatúra

Gemmell, Peter a Madhu Sudan. 1992. "Highly resilient correctors for polynomials." *Information processing letters* 43(4), s. 169–174.

Kapitola 19

Pekelne ťažké funkcie z veľmi ťažkých

Pripomeňme, že *veľmi ťažká* funkcia potrebuje exponenciálne veľký obvod, ak ju chceme spočítať na 99%, zato *pekelne ťažká* funkcia potrebuje exponenciálne veľký obvod, ak keď ju chceme spočítať čo i len na $\frac{1}{2} + \frac{1}{S}$ vstupoch.

Znie to až neuveriteľne. Existujú vôbec pekelne ťažké funkcie? Nie je to príliš silný predpoklad? V tejto kapitole ukážeme, že ak existujú veľmi ťažké funkcie, vieme z nich zostrojiť ešte ťažšie, dokonca pekelne ťažké funkcie.

A dokonca ťažkú funkciu zostrojíme veľmi jednoducho: Intuitívne pre každý problém môžu existovať vstupy (inštancie), ktoré sú jednoduché, ale aj vstupy, ktoré sú ťažké (predstavte si napríklad Sudoku – niektoré zadania majú veľa predvyplnených políčok a tie zvyšné vieme doplniť jednoduchými úvahami, iné zadania môžu byť ťažké a musíme skúšať rôzne možnosti). To, čo chceme dosiahnuť, je, aby väčšia časť vstupov bola „ťažká“. Majme teda funkciu f a vstup x. Rozdeľme ho na dve polovice $x = x_1x_2$ a uvažujme funkciu

$$g(x_1x_2) = f(x_1) \oplus f(x_2).$$

Ak chceme vypočítať $g(x_1x_2)$, musíme správne vypočítať aj x_1 aj x_2 (alebo obe nesprávne). Opäť intuitívne: ak je aspoň jeden zo vstupov x_1 alebo x_2 ťažký, potom spočítať $f(x_1) \oplus f(x_2)$ je ťažké. Ak má f napríklad $1/10$ ťažkých vstupov, potom g bude mať $2 \cdot 1/10 - 1/100$ ťažkých vstupov (takmer dvakrát viac).

Táto konštrukcia sa dá zovšeobecniť: definujme

$$f^{\oplus k}(x_1, \ldots, x_k) = \bigoplus_i f(x_i).$$

Yaova XOR lema hovorí, že

ak f je veľmi ťažká, tak $f^{\oplus k}$ je pekelne ťažká.

Presnejšie:

Lema 19.1 (Yaova XOR lema (Impagliazzo, 1995)). *Nech $f : \{0,1\}^n \to \{0,1\}$; ak je ťažké (treba exponenciálne veľký obvod) spočítať funkciu f na 99%, tak $f^{\oplus 1000}$ je ťažké spočítať čo i len na 50.01% vstupov a $f^{\oplus \mathsf{poly}(n)}$ je ťažké spočítať čo i len na $50 + 1/\exp(n)$% vstupov.*

Presnejšie, pre $\rho < 1$ a pre $\varepsilon > 2\rho^k$ je

$$\mathsf{H}^{1/2+\varepsilon}_{\mathrm{avg}}(f^{\oplus k}) \geq \mathsf{poly}(\varepsilon, 1/n) \cdot \mathsf{H}^{\rho}_{\mathrm{avg}}(f).$$

Napríklad pre $\rho = 0.99$ a $k = 1000$ je $2\rho^k < 0.01\%$.

Dôkaz tejto lemy naozaj postupuje podľa intuície vyššie, že $f^{\oplus k}$ má oveľa viac „ťažkých vstupov“ ako f. Zastavme sa však ešte pri spojení „ťažký vstup“, čo je nepresné – žiadny jeden konkrétny vstup nie je ťažký, pretože sa dá „vypočítať“ triviálnym obvodom, ktorý má správnu odpoveď zadrátovanú. Ťažká teda môže byť len množina dostatočne veľa vstupov. Avšak predstavme si, že zoberieme ťažký problém a „vyhodíme“ „ľahké“ vstupy. Ak sa obmedzíme len na ostávajúce „ťažké“ vstupy .....

Presne toto hovorí tzv. hardcore lema, alebo Lema o ťažkom jadre.

Hovoríme, že S je (δ-)veľká množina, ak má aspoň $\delta 2^n$ prvkov. Každej množine prirodzene prislúcha uniformná distribúcia U_S, kde prvky S majú pravdepodobnosť $1/|S|$ a ostatné nula. Predstavme si, že máme k veľkých množín $S_1, \ldots, S_k$ a uvažujme nasledovný postup: najskôr si podľa nejakej distribúcie α vyberieme jednu množinu S_i a z nej potom vyberieme náhodný prvok. Dostaneme tak distribúciu $\alpha_1 U_{S_1} + \alpha_2 U_{S_2} + \cdots + \alpha_k U_{S_k}$, mix uniformných distribúcií na veľkých množinách.

D je δ-distribúcia, ak je mix uniformných distribúcii na δ-veľkých množinách. D je δ-distribúcia, ak $D(x) \leq 1/\delta 2^n$ pre každé x.

Lema 19.2 (Hardcore lema (Impagliazzo, 1995)). *Ak f je veľmi ťažká, tak existuje množina H veľkosti $\frac{1}{100}2^n$ taká, že f je pekelne ťažká na H.*

Presnejšie: existuje množina H veľkosti $\delta 2^n$ taká, že žiadny obvod C menší ako $\varepsilon^2/100n \cdot \mathsf{H}^{1-\delta}_{\mathrm{avg}}(f)$ nedokáže spočítať f na viac ako $1/2+\varepsilon$-tine vstupov z H:

$$\Pr_{x \in_R H}[C(x) = f(x)] \leq 1/2 + \varepsilon.$$

Poďme si obe lemy dokázať.

19.1 Yaova XOR lema

Späť k Yaovej XOR leme:

$$f^{\oplus k}(x_1, \ldots, x_k) = \bigoplus_i f(x_i).$$

Pre $\delta > 0$, $\varepsilon > 2(1-\delta)^k$ je

$$\mathsf{H}^{1/2+\varepsilon}_{\mathrm{avg}}(f^{\oplus k}) \geq \underbrace{\frac{\varepsilon^2}{400n} \cdot \underbrace{\mathsf{H}^{1-\delta}_{\mathrm{avg}}(f)}_{S}}_{S'}.$$

■ **Dôkaz.** Ukážeme si dôkaz pre $k = 2$, zovšeobecnenie prenecháme čitateľovi. Sporom: Nech H je ťažké jadro, množina veľkosti $\delta 2^n$ a G je jej doplnok $G = \{0,1\}^n - H$. Predpokladajme, že existuje malý obvod C počítajúci $f^{\oplus 2}$ s pravdepodobnosťou $\geq 1/2 + \varepsilon$. Ukážeme, že z obvodu pre $f^{\oplus 2}$ vieme vyrobiť malý obvod pre f na ťažkom jadre H.

Pravdepodobnosť, že $C(x) = f^{\oplus 2}(x)$ môžeme rozložiť na štyri možnosti, podľa toho, či x_1 a x_2 patria do H alebo G a keďže $|H| = \delta 2^n$:

$$\begin{aligned}\Pr_{x\in\{0,1\}^{2n}}[C(x) = f^{\oplus 2}(x)] &= \Pr[x_1x_2 \in GG] \cdot \Pr[C(x) = f^{\oplus 2}(x) \mid x_1x_2 \in GG] \\ &+ \Pr[x_1x_2 \in HG] \cdot \Pr[C(x) = f^{\oplus 2}(x) \mid x_1x_2 \in HG] \\ &+ \Pr[x_1x_2 \in GH] \cdot \Pr[C(x) = f^{\oplus 2}(x) \mid x_1x_2 \in GH] \\ &+ \Pr[x_1x_2 \in HH] \cdot \Pr[C(x) = f^{\oplus 2}(x) \mid x_1x_2 \in HH]\end{aligned}$$

Ak pre podmnožinu $\mathcal{D} \subseteq \{0,1\}^{2n}$ označíme $P_{\mathcal{D}} = \Pr_{x\in_R\mathcal{D}}[C(x) = f^{\oplus 2}(x)] = \Pr_{x\in\{0,1\}^{2n}}[C(x) = f^{\oplus 2}(x) \mid x \in \mathcal{D}]$, túto istú rovnosť vieme napísať jednoduchšie:

$$P_{\{0,1\}^{2n}} = \underbrace{(1-\delta)^2}_{<\varepsilon/2}\underbrace{P_{G^2}}_{\leq 1} + (1-\delta)\delta P_{GH} + \delta(1-\delta)P_{HG} + \delta^2 P_{H^2}$$

Podľa nášho predpokladu $P_{\{0,1\}^{2n}} \geq 1/2 + \varepsilon$ a $(1-\delta^2) < \varepsilon/2$ a ak pravdepodobnosť P_{G^2} odhadneme zhora 1 (nech C počíta f mimo jadra čo aj so 100% úspečnosťou), dostaneme

$$1/2 + \varepsilon/2 \leq (1-\delta)\delta P_{GH} + \delta(1-\delta)P_{HG} + \delta^2 P_{H^2}$$

Súčet koeficientov na pravej strane je menej ako 1 a preto z priemerovacieho princípu vyplýva, že aspoň jeden člen $P_{\mathcal{D}}$ musí byť viac ako $1/2 + \varepsilon/2$. Predpokladajme napríklad, že $P_{HG} > 1/2 + \varepsilon/2$, teda

$$\Pr_{x_1\in_R H, x_2\in_R G}[C(x_1,x_2) = f(x_1) \oplus f(x_2)] > 1/2 + \varepsilon/2.$$

Opäť, podľa priemerovacieho princípu musí existovať konkrétne x_2 také, že

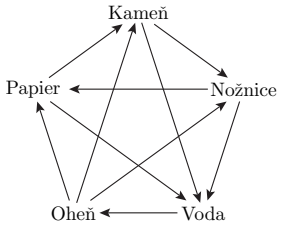
$$\Pr_{x_1\in_R H}[\underbrace{C(x_1,x_2) \oplus f(x_2)}_{D(x_1)} = f(x_1)] > 1/2 + \varepsilon/2.$$

Potom ale môžeme zobrať toto jedno konkrétne x_2 a konkrétnu hodnotu $f(x_2)$ a zadrátovať ju do C. Dostaneme tak obvod D veľkosti S' (počítajúci $x_1 \mapsto C(x_1,x_2) \oplus f(x_2)$ so zadrátovaným $x_2, f(x_2)$), ktorý počíta f na H s pravdepodobnosťou $> 1/2 + \varepsilon/2$, čo je spor s hardcorovosťou H. □

19.2 Malá odbočka do teórie hier

Skôr ako pustíme do dôkazu hardcore lemy, spravíme malú odbočkou do teórie hier – budeme ju potrebovať v dôkaze.

Vezmime si napríklad hru kameň–papier–nožnice, alebo, nech je to zaujímavejšie, kameň–papier–nožnice–oheň–voda, kde oheň vyhrá nad všetkým okrem vody a voda prehrá so všetkým okrem ohňa. Na obrázku 19.1 je táto hra zobrazená vo forme grafu a matice výhier. Otázka samozrejme znie, ako máme hrať? Aká stratégia je optimálna? (Vedeli by ste ju nájsť?)



(a) Graf „čo porazí čo“

	K	P	N	O	V
K	0	−1	+1	−1	+1
P	+1	0	−1	−1	+1
N	−1	+1	0	−1	+1
O	+1	+1	+1	0	−1
V	−1	−1	−1	+1	0

(b) Matica výhier

Obr. 19.1: Hra kameň–papier–nožnice–oheň–voda.

Druhý príklad je hra Morra, ktorú hrávali už starovekí Gréci či Rimania. Má viacero variácií a môžu ju hrať aj viacerí hráči, my však budeme uvažovať nasledovnú verziu: Hrajú dvaja hráči, ktorí naraz ukážu 1 až 5 prstov na ruke a zvolajú číslo (od 2 do 10). Vyhrá ten hráč, ktorý uhádne súčet ukázaných prstov na oboch rukách – tento počet je zároveň jeho výhra. Ak ani jeden netrafí, alebo ak obaja zvolajú rovnaké číslo, je remíza.

Napríklad ak Alica ukáže 3 prsty a povie „7“ a Bob ukáže 4 prsty a povie „6“, Alica vyhrala (lebo $3 + 4 = 7$) a Bob jej musí zaplatiť 7 mincí. Takáto hra sa dá reprezentovať maticou M veľkosti 25×25:

$$M_{(i,k),(j,\ell)} = \begin{cases} +k & \text{ak } i + j = k \neq \ell \\ -\ell & \text{ak } i + j = \ell \neq k \\ 0 & \text{inak,} \end{cases}$$

kde index $(i, k), (j, \ell)$ znamená, že Alica ukázala i prstov a zvolala k a Bob ukázal j prstov a zvolal ℓ. Opäť rovnaká otázka: vedeli by ste nájsť optimálnu stratégiu, ako hrať?

Vo všeobecnosti majme maticu $\mathbf{A}$ a uvažujme nasledovnú hru: Alica si zvolí riadok i, Bob si zvolí stĺpec j a to je celé – koniec hry. Hrá sa o peniaze a A_{ij} je výhra/prehra z pohľadu Alice, t.j., ak A_{ij} je kladné, vyhrala Alica a Bob jej vyplatí A_{ij}; naopak, ak je A_{ij} záporné, vyhral Bob a Alica mu vyplatí $|A_{ij}|$. Takéto hry, kde jeden hráč získa presne toľko, čo druhý hráč stratí, sa volajú hry s nulovým súčtom.

A čo si predstavujeme pod odpoveďou na otázku „čo mám robiť“, „ako mám hrať“? Očakávame pravdepodobnostné rozdelenie, ktoré pre každý možný

(a) Hra Morra, David Allan, 1765

	$(1,2)$	$(1,3)$	$(2,3)$	$(2,4)$
$(1,2)$		$+2$	-3	
$(1,3)$	-2			$+3$
$(2,3)$	$+3$			-4
$(2,4)$		-3	$+4$	

Optimálna stratégia:
$(1,2)$ s pravdepodobnosťou
$4/7 \approx 57\%$ a
$(2,4)$ s pravdepodobnosťou
$3/7 \approx 43\%$.

(Nikdy sa neoplatí povedať „3“.)

(b) Matica hry, ak by sme sa obmedzili iba na 2 prsty.

Obr. 19.2: Hra Morra.

ťah povie s akou pravdepodobnosťou ho zvoliť. Takáto pravdepodobnostná distribúcia nad riadkami pre Alicu a stĺpcami pre Boba sa v teórii hier nazýva tiež zmiešaná stratégia (na rozdiel od deterministickej „rýdzej“ stratégie, kde si Alica a Bob môžu vybrať iba jeden fixný riadok/stĺpec). Napríklad v hre kameň–papier–nožnice je najlepšie zvoliť každý ťah s pravdepodobnosťou 1/3. A optimálna stratégia v hre Morra je:

$$p_{(1,2)} = 56/190 \approx 29.5\%,$$
$$p_{(2,4)} = 42/190 \approx 22.1\%,$$
$$p_{(3,6)} = 35/190 \approx 18.4\%,$$
$$p_{(4,8)} = 30/190 \approx 15.8\%,$$
$$p_{(5,10)} = 27/190 \approx 14.2\%,$$

kde $p_{(i,j)}$ je pravdepodobnosť, že ukážem i prstov a zvolám j. Zaujímavé je, že sa neoplatí zvolať iné číslo ako dvojnásobok počtu prstov, ktoré ukážem. (Vedeli by ste zdôvodniť, prečo?)

Označme $\mathbf{p}, \mathbf{q}$ pravdepodobnostné distribúcie nad riadkami/stĺpcami (zapísané ako stĺpcové vektory). Potom pravdepodobnosť, že hráči zvolia i-ty riadok a j-ty stĺpec je $p_i q_j$ a teda očakávaný bodový zisk Alice je

$$\begin{aligned}\mathrm{E}[\text{výhra Alice}] &= \sum_{i,j} \underbrace{\Pr[\text{Alica zvolí } i]}_{p_i} \cdot \underbrace{\Pr[\text{Bob zvolí } j]}_{q_j} \cdot A_{ij} \\ &= \sum_{i,j} p_i \cdot A_{ij} \cdot q_j = \mathbf{p}^T \mathbf{A} \mathbf{q}.\end{aligned}$$

Inými slovami, ak by Alica a Bob hrali opakovane, pričom by vždy volili náhodný riadok/stĺpec podľa distribúcii $\mathbf{p}$ a $\mathbf{q}$, potom v priemere by Alica vyhrala $\mathbf{p}^T \mathbf{A} \mathbf{q}$ (za jednu hru).

Ak by Alica poznala Bobovu stratégiu $\mathbf{q}$, potom si vie pre každý riadok spočítať priemernú výhru:

$$\text{E[výhra pre } i\text{-ty riadok]} = \sum_j \Pr[\text{Bob zvolí } j] \cdot A_{ij} = \mathbf{Aq}.$$

Stačilo by jej teda zvoliť riadok s maximálnou očakávanou výhrou. Naopak, ak by Bob poznal Alicinu stratégiu $\mathbf{p}$, zvolil by stĺpec $\mathbf{p}^T\mathbf{A}$ s minimálnou hodnotou. Teda ak by sme poznali stratégiu súpera, je jednoduché prísť s optimálnou proti-stratégiou (a tá je navyše deterministická). Inými slovami, označme $\mathbf{e}_i$ vektor, ktorý má na i-tej pozícii jednotku a všade inde nuly, potom

$$\max_{\mathbf{p}} \mathbf{p}^T\mathbf{Aq} = \max_i \mathbf{e}_i^T\mathbf{Aq} \qquad \text{a} \qquad \min_{\mathbf{q}} \mathbf{p}^T\mathbf{Aq} = \min_j \mathbf{p}^T\mathbf{Ae}_j.$$

Predstavme si, že pre každú Alicinu stratégiu $\mathbf{p}$ spočítame hodnotu $v_A(\mathbf{p}) = \min_{\mathbf{q}} \mathbf{p}^T\mathbf{Aq}$ – koľko získa, ak Bob zvolí optimálnu stratégiu proti $\mathbf{p}$. To znamená, ak Alica použije $\mathbf{p}$, má garantovanú výhru aspoň $v_A(\mathbf{p})$. Mohla by skúsiť túto hodnotu maximalizovať:

$$\text{nech } v_A = \max_{\mathbf{p}} v_A(\mathbf{p}) = \max_{\mathbf{p}}(\min_{\mathbf{q}} \mathbf{p}^T\mathbf{Aq}) = \max_{\mathbf{p}}(\min_j \mathbf{p}^T\mathbf{Ae}_j).$$

Pri takejto stratégii, označme ju $\tilde{\mathbf{p}}$, má garantovanú výhru aspoň v_A. Pri iných stratégiach to je (pri optimálnej hre súpera) menej (nanajvýš rovnako veľa).

Podobne by Bob mohol skúsiť pre každú svoju stratégiu $\mathbf{q}$ vypočítať optimálnu proti-stratégiu Alice a koľko pritom Alica získa: $v_B(\mathbf{q}) = \max_{\mathbf{p}} \mathbf{p}^T\mathbf{Aq}$ – túto hodnotu chce Bob minimalizovať:

$$\text{nech } v_B = \min_{\mathbf{q}} v_B(\mathbf{q}) = \min_{\mathbf{q}}(\max_{\mathbf{p}} \mathbf{p}^T\mathbf{Aq}) = \min_{\mathbf{q}}(\max_i \mathbf{e}_i^T\mathbf{Aq}).$$

Pri takejto stratégii, označme ju $\tilde{\mathbf{q}}$, Alica vyhrá najviac v_B. Pri iných Bobových stratégiach môže vyhrať aj viac.

Samozrejme, implicitne sme predpokladali, že Alica a Bob ťahajú naraz a nevedia, ako ten druhý potiahne – nepoznajú jeho stratégiu. A nie je ani jasné či optimálna stratégia vždy existuje (možno proti Bobovej stratégii $\mathbf{q}$ je najlepšia Alicina stratégia $\mathbf{p}$, proti ktorej je zase najlepšie použiť $\mathbf{q}'$, atď.). John von Neumann však dokázal, že optimálne stratégie existujú *vždy* a hráčom vôbec nepomôže, ani keď dopredu poznajú stratégiu protihráča. Skúste si rozmyslieť, že musí platiť

$$v_A \leq v_B$$

(to je pomerne jednoduché – totiž pre ľubovoľnú funkciu f platí $\max_x \min_y f(x,y) \leq \min_x \max_y f(x,y)$). Von Neumann však dokázal, že

$$v_A = v_A(\tilde{\mathbf{p}}) = \tilde{\mathbf{p}}^T\mathbf{A}\tilde{\mathbf{q}} = v_B(\tilde{\mathbf{q}}) = v_B.$$

Táto hodnota $v = \tilde{\mathbf{p}}^T\mathbf{A}\tilde{\mathbf{q}}$ sa volá *hodnota hry* $\mathbf{A}$. Ak Alica použije $\tilde{\mathbf{p}}$, má garantovanú výhru aspoň v a ak Bob použije $\mathbf{q}$, má garantovanú stratu[1] najviac v. Alici

[1] Samozrejme, ak $v < 0$, je to Alicina strata a Bobova výhra. Ak $v = 0$, hra je férová a pri optimálnej hre oboch súperov je očakávaná výhra/strata nulová.

ani Bobovi sa neoplatí hrať inak, pretože v tom prípade môžu pri optimálnej hre súpera získať menej/stratiť viac.

Veta 19.1 (Neumann (1928)). *Pre každú hru s nulovým súčtom danú maticou* $\mathbf{A}$ *platí*

$$\max_{\mathbf{p}}(\min_{\mathbf{q}} \mathbf{p}^T\mathbf{A}\mathbf{q}) = \min_{\mathbf{q}}(\max_{\mathbf{p}} \mathbf{p}^T\mathbf{A}\mathbf{q}).$$

Optimálna stratégia Alice je distribúcia $\tilde{\mathbf{p}}$*, ktorá maximalizuje ľavú stranu a optimáln stratégia Boba je distribúcia* $\tilde{\mathbf{q}}$*, ktorá minimalizuje pravú stranu.*

Len na okraj ešte prezraďme, že hodnota hry a optimálne stratégie sa dajú nájsť pomocou lineárneho programovania. Pripomeňme, že minimum = najväčšie dolné ohraničnie:

$$\min X = \max\{m \mid \forall x \in X : m \leq x\}.$$

Takže hodnotu $v_A(\mathbf{p}) = \min_j \mathbf{p}^T\mathbf{A}\mathbf{e}_j$ vieme vyjadriť ako najväčšie v také, že

$$(\mathbf{p}^T\mathbf{A})_j = \sum_i p_i A_{ij} \geq v \qquad \text{pre každé } j.$$

Výpočet minima sme tak nahradili sériou lineárnych nerovníc. Hodnotu v chceme maximalizovať cez všetky Alicine stratégie $\mathbf{p}$ – takže úlohu môžeme preformulovať nasledovne:

$$\begin{aligned} &\text{maximalizuj hodnotu } v, \\ &\text{za podmienok: } \sum_i p_i A_{ij} \geq v && \text{pre } j = 1, \ldots, n \\ &\text{a } \; p_1 + p_2 + \cdots + p_n = 1 && (\mathbf{p} \text{ je pravdepodobnostná} \\ &\qquad p_1, p_2, \ldots, p_n \geq 0 && \text{distribúcia}) \end{aligned}$$

alebo skrátene:

$$\begin{aligned} &\max v, \\ &\text{za podmienok: } \mathbf{p}^T\mathbf{A} \geq v \cdot \mathbf{1} \\ &\qquad \mathbf{p}^T\mathbf{1} = 1 \\ &\qquad \mathbf{p} \geq \mathbf{0} \end{aligned}$$

... a podobne pre Boba:

$$\begin{aligned} &\min v, \\ &\text{za podmienok: } \mathbf{A}\mathbf{q} \leq v \cdot \mathbf{1} \\ &\qquad \mathbf{1}^T\mathbf{q} = 1 \\ &\qquad \mathbf{q} \geq \mathbf{0} \end{aligned}$$

A to sú presne úlohy lineárneho programovania a tie sa dajú vyriešiť v polynomiálnom čase. Navyše tieto dva programy sú navzájom duálne a z teórie lineárneho programovania vyplýva, že majú rovnaké optimá – von Neumannova min-max veta vyplýva z vety o silnej dualite.

19.3 Ťažké jadro

Späť k Hardcore leme.

Na tejto leme je zaujímavé, že je uveriteľná alebo neuveriteľná podľa toho, z ktorej strany sa číta. Intuitívne je uveriteľné, že ak je nejaký problém ťažký a vyberieme z neho sadu len tých „ťažkých“ vstupov, dostaneme zapeklitý problém (je ťažké riešiť ho aj s pomerne malou úspešnosťou).

Pozrime sa však na tú istú lemu z druhej strany (na obmenu implikácie). Lema vraví:

$$\text{ak } \ f \text{ je veľmi ťažká} \Longrightarrow \exists \text{pekelne ťažké jadro } H,$$

tzn.

$$\text{ak } \begin{array}{c}\text{každý malý obvod}\\ \text{spočíta } f \text{ na } < 99\%\end{array} \Longrightarrow \exists \text{množina } H \ \begin{array}{c}\forall \text{malý obvod dokáže vyriešiť}\\ \text{len } < 50+\varepsilon\% \text{ vstupov z } H.\end{array}$$

Obmena:

$$\not\exists \text{pekelne ťažké jadro} \Longrightarrow \exists \text{malý obvod, ktorý počíta } f \text{ skoro všade},$$

tzn.

$$\text{ak } \forall \text{množinu } H \ \begin{array}{c}\exists \text{malý obvod } C, \text{ktorý vyrieši}\\ \geq 50+\varepsilon\% \text{ vstupov z } H\end{array} \Longrightarrow \begin{array}{c}\exists \text{malý obvod, ktorý}\\ \text{rieši } f \text{ na } \geq 99\%.\end{array}$$

Predstavme si, že učiteľ chce dať na konci školského roka písomku. Všetky možné otázky sú dopredu známe s tým, že na teste bude nejaká podmnožina z nich. Žiakov je veľmi veľa a sú leniví (alebo nestíhajú), takže sa nenaučia celé učivo, ale len si letmo prebehnú nejakú sadu otázok. Predstavme si, že na každú sadu otázok (pre každý test) máme nejakého „experta“, ktorý ten test napíše na 51% (čo je veľmi slabý výsledok, len o 1% lepšie ako úplná tipovačka). Veta o ťažkom jadre hovorí, že ak títo experti dajú hlavy dokopy, môžu test napísať na 99%(!) A to znie pomerne neuveriteľne.

Táto veta má teda veľmi praktickú motiváciu a aplikácie v teórii strojového učenia. Namiesto „žiak“ si dosaďte „klasifikátor“. Na trénovacích dátach si najskôr natrénujeme hordu pomerne slabých klasifikátorov. Vieme z nich vyskladať dobrý klasifikátor? Odpoveď je Áno, existujú rôzne metódy známe pod menom „boosting“.

Samozrejme, v teórii učenia potrebujeme možno silnejšie výsledky, možno pre konkrétne modely klasifikátorov a najmä potrebujeme efektívny postup, ako zo slabých klasifikátorov vyrobiť ten silný. My si v tejto kapitole ukážeme čo najjednoduchší dôkaz. Bude dosť nekonštruktívny (dokážeme iba existenciu obvodu, nie ako ho zostrojiť), ale to nám postačí.

■ **Dôkaz.** Uvažujme takúto hru:

- Alica zvolí množinu vstupov H veľkosti aspoň $\delta 2^n$
- Bob zvolí obvod C veľkosti $\leq S'$
- Alica zaplatí Bobovi sumu $v = \Pr_{x \in_R H}[C(x) = f(x)]$

Z nášho predpokladu vie Bob vždy vyhrať aspoň $v \geq 1/2 + \varepsilon$

$$\forall H \exists C : \Pr_{x \in_R H}[C(x) = f(x)] \geq 1/2 + \varepsilon$$

$$\Downarrow$$

$$\exists \mathcal{C} \forall H : \Pr_{C \in_R \mathcal{C}, x \in_R H}[C(x) = f(x)] \geq 1/2 + \varepsilon$$

Pomocou tejto distribúcie zostrojíme obvod, ktorý počíta f na skoro všetkých $(1 - \delta)$ vstupoch. Vyberme náhodne t obvodov $C_1, \ldots, C_t$ podľa distribúcie $\mathcal{C}$; výsledný obvod na vstupe vypočíta $C_i(x)$ pre $i = 1, \ldots, t$ a zvolí väčšinovú odpoveď. Veľkosť takéhoto obvodu C bude $t \times S' +$ dačo $< S$.

Nazvime reťazec x „zlý“, ak menej ako polovica $+\varepsilon$ obvodov dá na ňom správnu odpoveď:

$$x \text{ je zlý, ak } \Pr_{C \in_R \mathcal{C}}[C(x) = f(x)] < 1/2 + \varepsilon.$$

Zlých reťazcov je len málo, konkrétne menej ako $\delta 2^n$. V opačnom prípade by sme mohli vybrať množinu všetkých zlých reťazcov Z a na nej by $\Pr_{C \in_R \mathcal{C}, x \in_R Z}[C(x) = f(x)] < 1/2 + \varepsilon$, čo je spor.

Sústreďme sa teraz len na tie „dobré“ x, teda také, že náhodný obvod $C \in_R \mathcal{C}$ odpovie správne s pravdepodobnosťou aspoň $1/2 + \varepsilon$. Ale potom väčšinová odpoveď z t obvodov bude s vysokou pravdepodobnosťou správna – rovnako ako keď zvyšujeme úspešnosť pravdepodobnostných algoritmov opakovaním. Presnejšie pre $t = 50n/\varepsilon^2$ z Černofovej nerovnosti vyplýva, že

$$\Pr_{C_1, \ldots, C_t \in_R \mathcal{C}}[\textsc{Maj}(C_1(x), \ldots, C_t(x)) \neq f(x)] \ll 1/2^n \qquad \text{pre všetky dobré } x.$$

Podľa union bound je potom nenulová pravdepodobnosť, že C odpovie správne na všetkých dobrých x a teda existuje konkrétna voľba $C_1, \ldots, C_t$, existuje konkrétny obvod C, ktorý odpovie správne na všetkých dobrých vstupoch. A keďže zlých vstupov je menej ako $\delta 2^n$, obvod C odpovedá správne aspoň na $(1 - \delta)$-tine vstupov. Tak sme dokázali, že ak pre všetky množiny veľkosti $\delta 2^n$ existuje obvod, ktorý počíta f na $(1/2 + \varepsilon)$-tine vstupov, tak $\mathsf{H}^{1-\delta}_{\text{avg}}(f) < S$. Alebo naopak, ak f je veľmi ťažká, tak obsahuje pekelne ťažké jadro.

□

Dôkazy pomocných tvrdení

Lema 19.3. *D je δ-distribúcia, tzn. mix uniformných distribúcii na množinách veľkosti aspoň $\delta 2^n$ práve vtedy, keď $D(x) \leq 1/(\delta 2^n)$ pre každé x.*

Lema 19.4. *Ak 2δ-distribúcia H je ε-ťažké jadro, potom existuje množina S veľkosti aspoň $\delta 2^n$, ktorá je 2ε-ťažké jadro.*

19.4 Naozaj pekelne ťažké funkcie

Možno by sa zdalo, že už sme vyhrali: ak f je veľmi ťažká, potom $f^{\oplus \mathsf{poly}(n)}$ vieme spočítať len s exponenciálne malou výhodou. Problém je, že sa nám zároveň nafúkne veľkosť vstupu. Napríklad ak sa f nedá spočítať obvodom veľkosti $2^{O(n)}$, tak $f^{\oplus O(n)}$ sa nedá spočítať obvodom veľkosti $2^{O(n)}$, avšak vzhľadom na veľkosť vstupu $N = n^2$ je to len $2^{O(\sqrt{N})}$.

Toto stačí na slabší výsledok, že $\mathsf{BPP} \subseteq \mathsf{QuasiP}$ (ak existuje $f \in \mathsf{E}$, ktorá potrebuje obvod zložitosti 2^{n^γ}). Na *naozaj* pekelne ťažké funkcie potrebujeme naše konštrukcie ešte vylepšiť. Dá sa to urobiť viacerými spôsobmi:

Derandomizovaná XOR lema

Ako prví dokázali dôsledok $\mathsf{BPP} = \mathsf{P}$ Impagliazzo a Wigderson (1997). Ich nápad bol „derandomizovať“ XOR lemu! XOR lema vraví, že ak

$$x_1, x_2, \ldots, x_k$$

sú úplne náhodné, potom je ťažké spočítať $\bigoplus_i f(x_i)$. Na to však potrebujeme príliš veľa náhodnosti. Čo keby sme namiesto toho použili pseudonáhodný generátor? Vstup x by sme nedelili na menšie časti, ale považovali by sme ho za seed do nášho generátora. Ten by vygeneroval pseudonáhodné vstupy

$$G(x) = z_1, z_2, \ldots, z_k,$$

a úlohou by bolo spočítať $f^{\oplus G} \equiv \bigoplus_i f(z_i) = \bigoplus_i f(G(x)_i)$.

Chceli by sme taký generátor, ktorý dokáže vygenerovať výstup dĺžky povedzme $\Omega(n^2)$, pričom seed má dĺžku iba $O(n)$ a aby $f^{\oplus G}$ bola pekelne ťažká, t.j. aby výhoda klesala exponenciálne. Je trochu ironické, že pri derandomizácií pravdepodobnostných algoritmov chceme vytvoriť dobrý generátor z pekelne ťažkej funkcie a na ňu potrebujeme opäť derandomizovať. Uvedomme si však, že tentoraz potrebujeme PNG s trochu inými parametrami: 1.) potrebujeme „oklamať“ iba XOR lemu, nie všetky obvody danej veľkosti a 2.) stačí nám kvadratické predĺženie.

Dobrá správa je, že také generátory existujú, hoci vysvetlenie všetkých detailov je nad rámec tejto knihy. Hlavná myšlienka je založená na použití *expanderov* (pozri Hoory a spol. (2006)), čo sú grafy, ktoré na jednej strane majú veľmi málo hrán, zato sú veľmi dobre poprepájané. Presnejšie: majú konštantný stupeň d, ale každý rez má aspoň $\Omega(n)$ hrán. Ekvivalentne, ak si vezmeme ľubovoľnú množinu najviac $n/2$ vrcholov S, tak tieto vrcholy majú $\Omega(n)$ susedov mimo S. Takéto grafy majú dobré miešacie vlastnosti: ak začneme v nejakom vrchole a spravíme náhodnú prechádzku – t.j. vždy si náhodne vyberieme jedného z d susedov – už po pár krokoch sa môžeme ocitnúť v ľubovoľnom vrchole grafu, dokonca v každom vrchole budeme s takmer rovnakou pravdepodobnosťou! (Presnejšie: nech $P_t(x)$ je pravdepodobnosť, že po t krokoch skončím vo vrchole x. Potom rozdiel medzi P_t a uniformnou distribúciou $U(x) = 1/n$, $\sqrt{\sum_{x \in V}(P_t(x) - 1/n)^2}$, klesá exponenciálne rýchlo.) Inými slovami, ak začnem v ľubovoľnom vrchole,

spravím náhodnú prechádzku, pričom každých zopár krokov si zapíšem, kde som, tak výsledok sa bude celkom podobať na úplne náhodný výber vrcholov.

... vlastne začneme v náhodnom vrchole a zapíšeme si každý...

ak chceme t náhodných vrcholov, potrebujeme $t \cdot \log n$ bitov. ak namiesto toho použijeme náhodnú prechádzku, stačí nám $\log n + O(t)$ bitov.

Extrahovanie náhodnosti

Druhá možnosť je pozrieť sa na to, prečo Nisan-Wigdersonov generátor potrebuje pekelne ťažké funkcie a čo by sa stalo, keby sme použili len veľmi ťažkú. Intuívne, ak x je ťažký vstup, tak $f(x)$ je na nerozoznanie od náhodného bitu, takže ak je f pekelne ťažká, skoro všetky vstupy sú ťažké a výstup $NW^f_{\mathcal{I}}$ vyzerá náhodne. Ak je f iba veľmi ťažká, stále má ťažké jadro – nejaký zlomok δ vstupov, ktoré sú extrémne ťažkých. Z toho vyplýva, že očakávame aspoň δ bitov $NW^f_{\mathcal{I}}$ generátora vyzerá náhodne. Aj s veľmi ťažkou funkciou teda vieme vygenerovať postupnosť, ktorá síce nie je úplne náhodná, ale má v sebe nejakú náhodnosť. Potrebovali by sme len túto náhodnosť akosi „vyextrahovať“.

Extrahovanie náhodnosti je mimochodom problém zaujímavý sám o sebe. Asi najstarší príklad je od von Neumanna: ako vygenerovať náhodný bit, ak máte iba mincu, kde hlava a znak nepadá s rovnakou pravdepodobnosťou? Ide tiež o veľmi praktický problém: Linuxový generátor `/dev/random` sleduje aktivitu myši, klávesnice, disku a rôzne interrupty, pričom predpokladá, že takéto udalosti síce nie sú náhodné, ale obsahujú „istú dávku náhodnosti“ – entropiu. Časy a popisy udalostí zhromažďuje a snaží sa odhadovať ich entropiu a extrahovať z nich náhodnosť. Na rozdiel od príkladu s mincou, kde sú jednotlivé hody nezávislé, `/dev/random` sa musí vysporiadať aj so závislosťami (napríklad pozícia myši závisí od predošlej polohy; ak niekto píše text, symboly vyťukané na klávesnici budú závislé, napríklad po dvoch spoluhláskach pôjde veľmi často samohláska; špecifické závislosti budú závisieť od jazyka/témy toho textu; časy medzi stlačeniami klávesov sú tiež závislé).

Ako mieru náhodnosti definujeme *min-entropiu* náhodnej premennej X ako $H_\infty(X) = \min_a 1/\Pr[X = a]$. Hovoríme, že $G : \{0,1\}^d \to \{0,1\}^m$ je (k, S, ε)-*generátor pseudoentropie*, ak existuje distribúcia D na $\{0,1\}^m$ s min-entropiou aspoň k taká, že žiadny obvod veľkosti S nedokáže odlíšiť D od $G(U_d)$ s výhodou viac ako ε.

Funkcia $\textsc{Ext} : \{0,1\}^m \times \{0,1\}^d \to \{0,1\}^n$ je (k, ε)-*extraktor*, ak pre každú distribúciu D na $\{0,1\}^m$ s min-entropiou aspoň k je štatistický rozdiel $\textsc{Ext}(D, U_d)$ a U_n najviac ε.

(Sudan a spol., 2001)

Zoznamové dekódovanie

Pravdepodobnostný algoritmus s orákulom A je lokálny zoznamový dekodér pre kód $C : \Sigma^k \to \Sigma^m$ s polomerom r, ak pre každé $y \in \Sigma^m$, A^y vypíše na výstup ℓ pravdepodobnostných algoritmov s orákulom $D_1, \ldots, D_\ell$ takých, že pre každý

reťazec x, ktorého kód je blízko y, $\Delta(C(x), y) \leq r$, s pravdepodobnosťou aspoň 3/4 existuje j také, že $\Pr[D_j^y(i) = x_i] \geq 3/4]$ $(\forall i)$.

Veta 19.2 (Goldreich-Levin 1989). *Nech $H : \{0,1\}^k \to \{0,1\}^{2^k}$ je Hadamardov kód. Existuje algoritmus, ktorý na vstupe y, ε beží v čase $\mathsf{poly}(k/\varepsilon)$ a s vysokou pravdepodobnosťou dá na výstupe zoznam všetkých x takých, že $\Delta(C(x), y) \leq 1/2 - \varepsilon$.*

Nech $x \in \{0,1\}^k$; definujme $L_x(a) = a^T x$. Potom funkcia L_x (respektíve tabuľka hodnôt L_x pre všetky a) je Hadamardov kód reťazca x. Potom existuje algoritmus, ktorý na vstupe ε, s prístupom k funkcií $g : \{0,1\}^k \to \{0,1\}$ vypíše zoznam $O(1/\varepsilon^2)$ správ x, pre ktoré je $\Delta(L_x, g) \leq 1/2 - \varepsilon$. Respektíve, pravdepodobnosť, že x s takouto vlastnosťou sa objaví na zozname je aspoň 3/4.

....

Lokálne δ-zoznamové dekódovanie pre SOK E je dvojica pravdepodobnostných algoritmov s orákulom (D_1, D_2) takých, že pre každé g a $\hat{f} = E(f)$ δ-blízko g platí: S pravdepodobnosťou aspoň 2/3 D_1^g vygeneruje $(a_1, ..., a_s)$ také, že existuje i: $\forall x : \Pr[D_2^g(x, a_i) = f(x)] \geq 2/3$.

Ak E je SOK s lokálnym δ-zoznamovým dekódovaním (D_1, D_2), kde D_2 beží v čase t a f je ťažká pre obvody veľkosti S, tak $\hat{f} = E(f)$ sa nedá vypočítať obvodmi veľkosti S/t

19.5 BPP a polynomiálna hierarchia

Na záver si ukážme ešte jeden jednoduchý dôsledok: V kapitole 7 sme dokázali Sipser-Gácsovu vetu, podľa ktorej BPP patrí pod druhú úroveň polynomiálnej hierarchie:

$$\mathsf{BPP} \subseteq \Sigma_2^{\mathsf{P}},$$

teda BPP vieme derandomizovať za cenu dvoch alternácií.

Tento výsledok pomerne jednoducho vypadne z teórie, ktorú sme doteraz vybudovali: BPP vieme derandomizovať pomocou dobrého pseudonáhodného generátora a ten vieme zostrojiť pomocou pekelne ťažkej funkcie. Σ_2^{P} je však dostatočne silná, aby takúto pekelne ťažkú funkciu našla – jednoducho ju nedeterministicky natipuje a paralelne overí všetky obvody, že ju nedokážu aproximovať.

Ukážeme si dokonca silnejší výsledok:

$$\mathsf{BPP} \subseteq \mathsf{ZPP}^{\mathsf{NP}}.$$

Pripomeňme, že $\mathsf{ZPP} \subseteq \mathsf{RP} \subseteq \mathsf{NP}$, takže zjavne $\mathsf{ZPP}^{\mathsf{NP}}$ je menšia trieda ako $\mathsf{NP}^{\mathsf{NP}} = \Sigma_2^{\mathsf{P}}$. Túto vetu ako prví dokázali Zachos a Heller (1986), nasleduje dôkaz od Nisan a Wigderson (1994).

Dôsledok 19.1 (Nisan a Wigderson (1994)). $\mathsf{BPP} \subseteq \mathsf{ZPP}^{\mathsf{NP}}$. *Inými slovami, ak by sme mali orákulum pre* SAT*, vedeli by sme ľubovoľný pravdepodobnostný algoritmus, ktorý sa mýli len s malou pravdepodobnosťou, premeniť na algoritmus, ktorý sa nikdy nemýli a skončí očakávane v polynomiálnom čase.*

■ **Dôkaz.** Ako sme ukázali, na derandomizáciu BPP stačí pekelne ťažká funkcia f na $O(\log n)$ bitoch, ktorá sa nedá vypočítať obvodom veľkosti $O(n^3)$.

Pre $\mathsf{BPP} \subseteq \Sigma_2^\mathsf{P}$ stačí najskôr nedeterministicky natipovať tabuľku hodnôt f (tá je len polynomiálne dlhá(!) keďže $2^{c \log n} = n^c$) a následne ko-nedeterministicky (paralelne) overiť, že žiadny malý obvod ju nevie aproximovať. Pre každý obvod C veľkosti $\leq k \cdot n^3$ prejdeme všetkých n^c vstupov a spočítame, na koľkých vstupoch sa C zhoduje s f. Ak je zhoda menšia ako $1/2 + 1/S$, zostrojíme $NW_\mathcal{I}^f$ a pomocou neho odsimulujeme BPP algoritmus (spustíme pre všetky seedy a vyberieme väčšinovú odpoveď). Ako vidíme, výsledný algoritmus by bol pomerne komplikovaný, no stále v polynomiálnom čase.

Pre $\mathsf{BPP} \subseteq \mathsf{ZPP}^\mathsf{NP}$ stačí zvoliť náhodnú funkciu a v coNP overiť, že je veľmi ťažká, keďže skoro všetky funkcie sú veľmi ťažké (počítací argument)! Pomocou Yaovej XOR lemy z nej vyrobíme pekelne ťažkú funkciu a ďalej postupujeme, ako sme popísali vyššie. □

Úlohy

- Jazyky v NP sú také, že ak by nám niekto povedal „dôkaz“ π, tak ho vieme overiť deterministicky v polynomiálnom čase. Uvažujme teraz randomizovanú verziu NP: keď dostaneme dôkaz π, pri overovaní môžeme použiť náhodné bity a slovo patrí do jazyka, ak akceptujeme $\geq 2/3$ prípadov a nepatrí, ak akceptujeme $\leq 1/3$ prípadov. Dalo by sa teda povedať, že „overovač“ je BPP-algoritmus.

  Ešte raz a formálnejšie: Nech $\mathcal{C}$ je trieda jazykov, pre ktoré existuje deterministický TS V (overovač), bežiaci v polynomiálnom čase od x taký, že

  - ak $x \in L$, tak $\exists \pi : \Pr_r[V(x, \pi, r) = 1] \geq 2/3$
  - ak $x \notin L$, tak $\forall \pi : \Pr_r[V(x, \pi, r) = 1] \leq 1/3$.

  Na $\mathcal{C}$ sa dá pozerať ako na randomizovanú verziu NP. $\mathcal{C}$ obsahuje aj NP aj BPP.

  Dokážte, že ak existuje $f \in \mathsf{E}$ a $\varepsilon > 0$ také, že $\mathsf{H}_{\text{avg}}(f)(n) \geq 2^{\varepsilon n}$, tak $\mathcal{C} = \mathsf{NP}$. Inými slovami, za predpokladu, že existujú veľmi ťažké funkcie, vieme triedu $\mathcal{C}$ derandomizovať.

Literatúra

Hoory, Shlomo, Nathan Linial, a Avi Wigderson. 2006. “Expander graphs and their applications.” *Bulletin of the American Mathematical Society* 43(4), s. 439–561.

Impagliazzo, Russell. 1995. “Hard-core distributions for somewhat hard problems.” In *Proceedings of IEEE 36th Annual Foundations of Computer Science*. IEEE, s. 538–545.

Impagliazzo, Russell a Avi Wigderson. 1997. "P = BPP if E requires exponential circuits: Derandomizing the XOR lemma." In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. s. 220–229.

Neumann, John von. 1928. "Zur Theorie der Gesellschaftsspiele." *Mathematische Annalen* 100(1), s. 295–320.

Nisan, Noam a Avi Wigderson. 1994. "Hardness vs randomness." *Journal of computer and System Sciences* 49(2), s. 149–167.

Sudan, Madhu, Luca Trevisan, a Salil Vadhan. 2001. "Pseudorandom generators without the XOR lemma." *Journal of Computer and System Sciences* 62(2), s. 236–266.

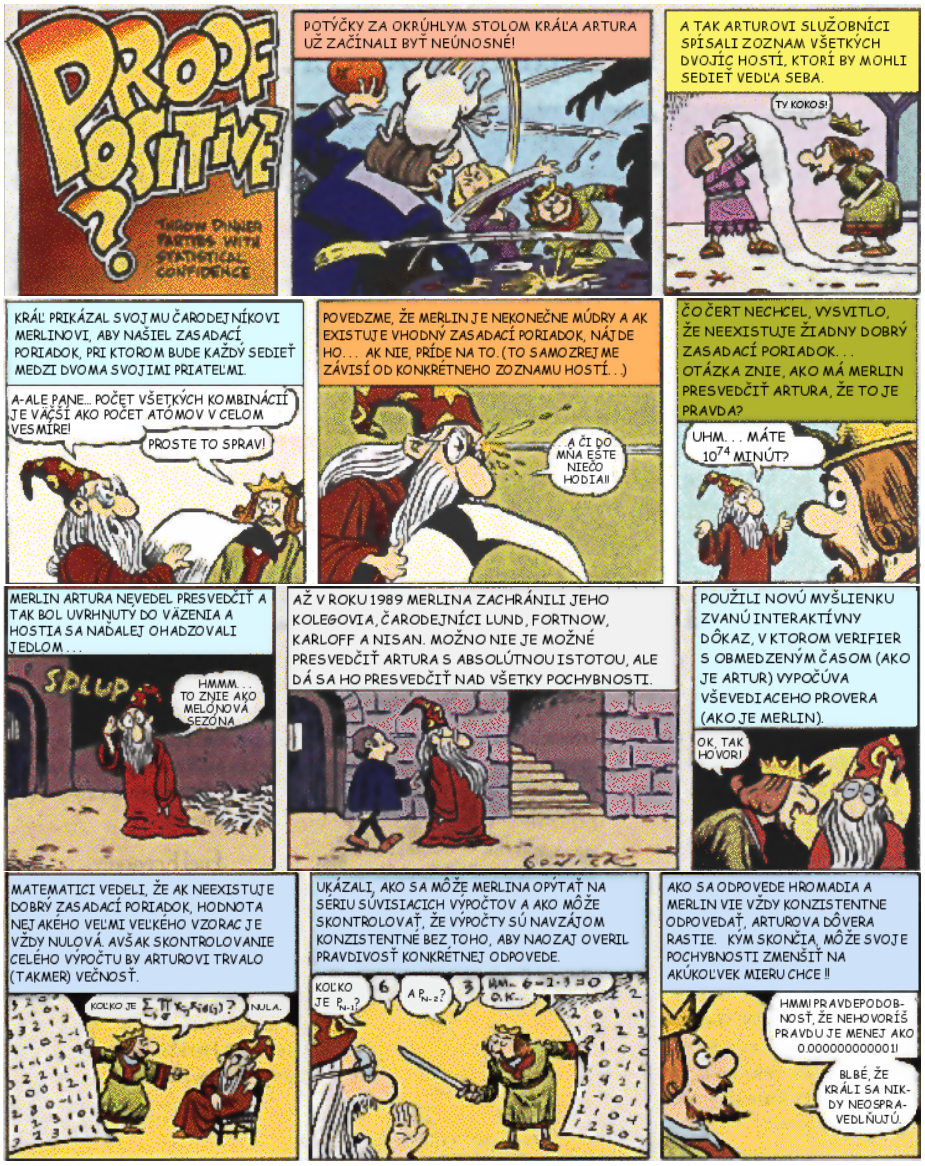
Zachos, Stathis a Hans Heller. 1986. "A decisive characterization of BPP." *Information and Control* 69(1-3), s. 125–135.

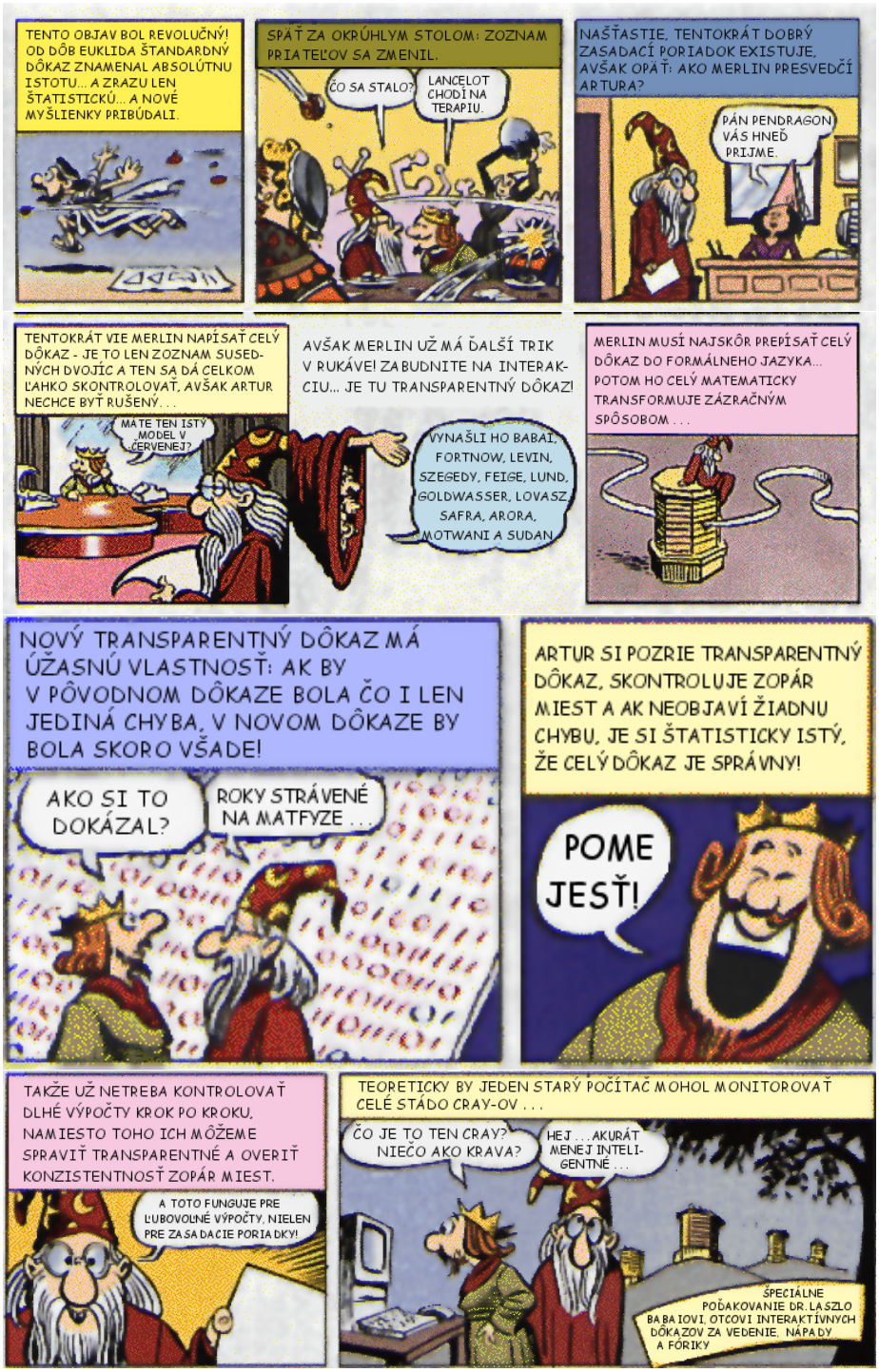
Časť VII

Interaktívne a pravdepodobnostne overiteľné dôkazy

Úvod

Asi najlepší úvod k tejto časti je nasledujúci komix:


PROOF POSITIVE?
THROW DINNER PARTIES WITH STATISTICAL CONFIDENCE
POTÝČKY ZA OKRÚHLYM STOLOM KRÁĽA ARTURA UŽ ZAČÍNALI BYŤ NEÚNOSNÉ!
A TAK ARTUROVI SLUŽOBNÍCI SPÍSALI ZOZNAM VŠETKÝCH DVOJÍC HOSTÍ, KTORÍ BY MOHLI SEDIEŤ VEDĽA SEBA.
TY KOKOS!
KRÁĽ PRIKÁZAL SVOJMU ČARODEJNÍKOVI MERLINOVI, ABY NAŠIEL ZASADACÍ PORIADOK, PRI KTOROM BUDE KAŽDÝ SEDIEŤ MEDZI DVOMA SVOJIMI PRIATEĽMI.
A-ALE PANE.. POČET VŠETKÝCH KOMBINÁCIÍ JE VÄČŠÍ AKO POČET ATÓMOV V CELOM VESMÍRE!
PROSTE TO SPRAV!
POVEDZME, ŽE MERLIN JE NEKONEČNE MÚDRY A AK EXISTUJE VHODNÝ ZASADACÍ PORIADOK, NÁJDE HO... AK NIE, PRÍDE NA TO. (TO SAMOZREJME ZÁVISÍ OD KONKRÉTNEHO ZOZNAMU HOSTÍ...)
A ČI DO MŇA EŠTE NIEČO HODIA!!
ČO ČERT NECHCEL, VYSVITLO, ŽE NEEXISTUJE ŽIADNY DOBRÝ ZASADACÍ PORIADOK... OTÁZKA ZNIE, AKO MÁ MERLIN PRESVEDČIŤ ARTURA, ŽE TO JE PRAVDA?
UHM... MÁTE 10^{74} MINÚT?
MERLIN ARTURA NEVEDEL PRESVEDČIŤ A TAK BOL UVRHNUTÝ DO VÄZENIA A HOSTIA SA NAĎALEJ OHADZOVALI JEDLOM...
SPLUP
HMMM... TO ZNIE AKO MELÓNOVÁ SEZÓNA
AŽ V ROKU 1989 MERLINA ZACHRÁNILI JEHO KOLEGOVIA, ČARODEJNÍCI LUND, FORTNOW, KARLOFF A NISAN. MOŽNO NIE JE MOŽNÉ PRESVEDČIŤ ARTURA S ABSOLÚTNOU ISTOTOU, ALE DÁ SA HO PRESVEDČIŤ NAD VŠETKY POCHYBNOSTI.
POUŽILI NOVÚ MYŠLIENKU ZVANÚ INTERAKTÍVNY DÔKAZ, V KTOROM VERIFIER S OBMEDZENÝM ČASOM (AKO JE ARTUR) VYPOČÚVA VŠEVEDIACEHO PROVERA (AKO JE MERLIN).
OK, TAK HOVORI
MATEMATICI VEDELI, ŽE AK NEEXISTUJE DOBRÝ ZASADACÍ PORIADOK, HODNOTA NEJAKÉHO VEĽMI VEĽKÉHO VZORCA JE VŽDY NULOVÁ. AVŠAK SKONTROLOVANIE CELÉHO VÝPOČTU BY ARTUROVI TRVALO (TAKMER) VEČNOSŤ.
KOĽKO JE ...?
NULA.
UKÁZALI, AKO SA MÔŽE MERLINA OPÝTAŤ NA SÉRIU SÚVISIACICH VÝPOČTOV A AKO MÔŽE SKONTROLOVAŤ, ŽE VÝPOČTY SÚ NAVZÁJOM KONZISTENTNÉ BEZ TOHO, ABY NAOZAJ OVERIL PRAVDIVOSŤ KONKRÉTNEJ ODPOVEDE.
KOĽKO JE P_{N-1}?
6
A P_{N-2}?
3
HM... $6-2\cdot 3=0$
O.K...
AKO SA ODPOVEDE HROMADIA A MERLIN VIE VŽDY KONZISTENTNE ODPOVEDAŤ, ARTUROVA DÔVERA RASTIE. KÝM SKONČIA, MÔŽE SVOJE POCHYBNOSTI ZMENŠIŤ NA AKÚKOĽVEK MIERU CHCE !!
HMM! PRAVDEPODOBNOSŤ, ŽE NEHOVORÍŠ PRAVDU JE MENEJ AKO 0.0000000000001!
BLBÉ, ŽE KRÁLI SA NIKDY NEOSPRAVEDLŇUJÚ.


TENTO OBJAV BOL REVOLUČNÝ! OD DÔB EUKLIDA ŠTANDARDNÝ DÔKAZ ZNAMENAL ABSOLÚTNU ISTOTU... A ZRAZU LEN ŠTATISTICKÚ... A NOVÉ MYŠLIENKY PRIBÚDALI.
SPÄŤ ZA OKRÚHLYM STOLOM: ZOZNAM PRIATEĽOV SA ZMENIL.
ČO SA STALO?
LANCELOT CHODÍ NA TERAPIU.
NAŠŤASTIE, TENTOKRÁT DOBRÝ ZASADACÍ PORIADOK EXISTUJE, AVŠAK OPÄŤ: AKO MERLIN PRESVEDČÍ ARTURA?
PÁN PENDRAGON VÁS HNEĎ PRIJME.
TENTOKRÁT VIE MERLIN NAPÍSAŤ CELÝ DÔKAZ - JE TO LEN ZOZNAM SUSEDNÝCH DVOJÍC A TEN SA DÁ CELKOM ĽAHKO SKONTROLOVAŤ. AVŠAK ARTUR NECHCE BYŤ RUŠENÝ...
MÁTE TEN ISTÝ MODEL V ČERVENEJ?
AVŠAK MERLIN UŽ MÁ ĎALŠÍ TRIK V RUKÁVE! ZABUDNITE NA INTERAKCIU... JE TU TRANSPARENTNÝ DÔKAZ!
VYNAŠLI HO BABAI, FORTNOW, LEVIN, SZEGEDY, FEIGE, LUND, GOLDWASSER, LOVASZ, SAFRA, ARORA, MOTWANI A SUDAN.
MERLIN MUSÍ NAJSKÔR PREPÍSAŤ CELÝ DÔKAZ DO FORMÁLNEHO JAZYKA... POTOM HO CELÝ MATEMATICKY TRANSFORMUJE ZÁZRAČNÝM SPÔSOBOM...
NOVÝ TRANSPARENTNÝ DÔKAZ MÁ ÚŽASNÚ VLASTNOSŤ: AK BY V PÔVODNOM DÔKAZE BOLA ČO I LEN JEDINÁ CHYBA, V NOVOM DÔKAZE BY BOLA SKORO VŠADE!
AKO SI TO DOKÁZAL?
ROKY STRÁVENÉ NA MATFYZE...
ARTUR SI POZRIE TRANSPARENTNÝ DÔKAZ, SKONTROLUJE ZOPÁR MIEST A AK NEOBJAVÍ ŽIADNU CHYBU, JE SI ŠTATISTICKY ISTÝ, ŽE CELÝ DÔKAZ JE SPRÁVNY!
POME JEŠŤ!
TAKŽE UŽ NETREBA KONTROLOVAŤ DLHÉ VÝPOČTY KROK PO KROKU, NAMIESTO TOHO ICH MÔŽEME SPRAVIŤ TRANSPARENTNÉ A OVERIŤ KONZISTENTNOSŤ ZOPÁR MIEST.
A TOTO FUNGUJE PRE ĽUBOVOLNÉ VÝPOČTY, NIELEN PRE ZASADACIE PORIADKY!
TEORETICKY BY JEDEN STARÝ POČÍTAČ MOHOL MONITOROVAŤ CELÉ STÁDO CRAY-OV...
ČO JE TO TEN CRAY? NIEČO AKO KRAVA?
HEJ... AKURÁT MENEJ INTELIGENTNÉ...
ŠPECIÁLNE POĎAKOVANIE DR. LÁSZLÓ BABAIOVI, OTCOVI INTERAKTÍVNYCH DÔKAZOV ZA VEDENIE, NÁPADY A FÓRIKY

Kapitola 20

Artur-Merlinove hry

20.1 Rôznofarebné ponožky a grafový neizomorfizmus

Začnime zopár príkladmi.

Predstavme si, že máme farboslepého kamaráta Fera a dve úplne rovnaké ponožky (čo sa týka veľkosti, tvaru, materiálu), ale rôznej farby. My jednoducho vidíme, že tie ponožky majú rôznu farbu, ale farboslepý kamarát to nevidí.

Vedeli by sme ho presvedčiť, že tie farby sú rôzne?

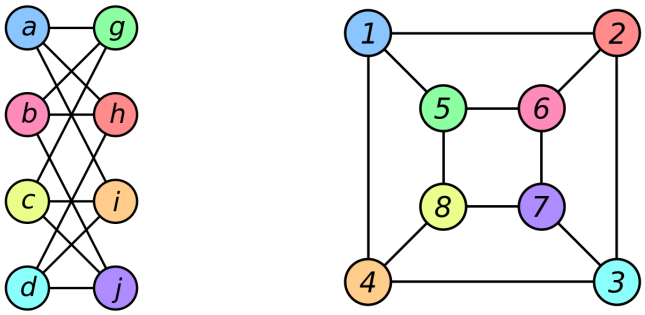
Tu je jednoduchý spôsob, ako sa to dá: povedzme kamarátovi, aké farby vidíme a nechajme ho, nech si túto informáciu poznačí niekde bokom, tak, aby sme to nevideli. Následne budeme opakovať nasledujúci pokus: Fero vytiahne *náhodnú* ponožku a spýta sa nás, ktorá to je. Našu odpoveď potom skontroluje, podľa toho, čo si poznačil.

Ak sú ponožky naozaj rôznofarebné, nie je problém pri každom pokuse správne odpovedať. Naopak, ak majú ponožky rovnakú farbu a teda sú neodlíšiteľné, pri každom pokuse máme iba 50% šancu, že trafíme správnu farbu. Ak teda pokus zopakujeme n-krát, pravdepodobnosť, že by sme nášho farboslepého kamaráta dokázali okabátiť je len $1/2^n$.

Ešte zaujímavejšie je, že okrem toho, že sme Fera presvedčili, že ponožky majú rôznu farbu bez toho, aby Fero vedel *prečo*. Bez toho, aby ich Fero vedel rozlišovať. Je to veľmi odlišný typ dôkazu napríklad od toho, keby sme chceli Fera presvedčiť, že 17 461 573 nie je prvočíslo. Prečo? Pretože $17\,461\,573 = 5479 \times 3187$. Aha, povie si Fero, keď súčin overí. Odteraz však vie Fero hocikoho iného presvedčiť, že 17 461 573 nie je prvočíslo – stačí, aby im ukázal dôkaz: $17\,461\,573 = 5479 \times 3187$. Avšak Fero by nedokázal teraz zobrať ponožky a rovnakým spôsobom presvedčiť svojho farboslepého kamaráta Fabiána, že sú rôznofarebné. Takýto dôkaz sa volá bezznalostný (zero knowledge).

Iný príklad: Grafový neizomorfizmus.

Predstavme si, že máme dva grafy a nevieme, či sú izomorfné (rovnaké až na premenovanie vrcholov). Zisťovať, či sú dva grafy izomorfné je vo všeobecnosti ťažký problém. V súčasnosti neexistuje polynomiálny algoritmus, ktorý by ho riešil, hoci sa nepredpokladá, že by problém bol NP-ťažký a existuje kvázipolynomiálny algoritmus (v čase $n^{\mathsf{polylog}(n)}$).



(a) (b)

Obr. 20.1: Dané dva grafy sú v skutočnosti len veľmi odlišne vyzerajúce nakreslenia toho istého grafu.

Predstavme si ďalej, že máme kamaráta Merlina, ktorý je mocný čarodej (alebo má rýchly superpočítač, na ktorom dokáže riešiť grafový izomorfizmus). Merlin tvrdí, že naše konkrétne dva grafy *nie sú* izomorfné. Vedel by nám to však rýchlo dokázať?

Ak by grafy boli izomorfné, jednoducho overiteľný dôkaz je izomorfizmus popísať, jednoducho vymenovať, ktoré vrcholy sú ekvivalentné. Avšak čo v prípade, že grafy nie sú izomorfné? Nevieme, či vždy existuje polynomiálne dlhý dôkaz, ktorý sa dá overiť v polynomiálnom čase. Inými slovami, grafový izomorfizmus je v NP, ale nevieme, či aj v coNP. Respektíve komplementárny problém, grafový neizomorfizmus je v coNP, ale nevieme, či je aj v NP.

Napriek tomu si ukážeme, že existuje jednoduchý interaktívny protokol, ktorým sa vieme presvedčiť, že $G_1 \not\equiv G_2$. Myšlienka je podobná ako pri ponožkách:

Veta 20.1 (Goldreich, Micali, Wigderson). *Existuje interaktívny protokol, podľa ktorého dokáže dokazovateľ overovateľa vždy presvedčiť, že dva grafy nie sú izomorfné. Ak však dané grafy sú izomorfné, šanca, že presvedčí je len 50% a dá sa ľubovoľne zmenšiť opakovaním protokolu. Grafový neizomorfizmus patrí do* IP[2].

■ **Dôkaz.** Dané sú grafy G_1, G_2; zvolíme náhodnú permutáciu π a náhodne zvolíme prvý alebo druhý graf. Dokazovateľovi dáme $H = \pi(G_i)$; ak $G_1 \not\equiv G_2$, buď je $H \equiv G_1$ alebo $H \equiv G_2$, takže dokazovateľ nemá problém zistiť, ktorý graf sme vybrali. V opačnom prípade $H \equiv G_1 \equiv G_2$ a dokazovateľovi neostáva nič iné ako si tipnúť. □

20.2 Súkromná a verejná minca

Všimnite si, že predchádzajúci protokol podstatne využíval fakt, že Artur si tajne hádzal mincou a Merlin nevedel, čo mu padlo (Artur sa na základe hodov mincou rozhodol, ktorý graf/ponožku ukáže). Hovoríme, že tento protokol využíval *súkromnú mincu* a otázka znie: Vedeli by sme nájsť protokol, pri ktorom Artur Merlinovi odhalí všetky hody mincou?

Veta 20.2. *Existuje interaktívny protokol s verejnou mincou pre grafový neizomorfizmus. Tento problém patrí do* AM[2].

■ **Dôkaz.** Navrhneme protokol, vďaka ktorému budeme vedieť rozlišovať, či je nejaká množina S „malá“, alebo „veľká“ (násobne väčšia).

Predpokladajme najskôr, že G_1 aj G_2 sú tzv. asymetrické grafy – ak prečíslujeme vrcholy, dostaneme iný graf – a uvažujme množinu

$$S = \{H \mid H \sim G_1 \vee H \sim G_2\}.$$

Ak $G_1 \sim G_2$, tak $|S| = n!$, ale ak $G_1 \not\sim G_2$, potom $|S| = 2n!$.

Vo všeobecnosti sa môže stať, že množina $\mathrm{iso}(G) = \{H \mid H \sim G\}$ je menšia ako $n!$. Vezmime napríklad graf P_3, cestu 1–2–3. Ak prečíslujeme vrcholy tak, že vymeníme 1 a 3, dostaneme identický graf (takúto permutáciu voláme automorfizmus), ale ak zmeníme číslo izolovaného vrcholu, dostaneme iný graf, takže $|\mathrm{iso}(P_3)| = 3$. Ak si vezmeme cyklus C_3, pri každej permutácii vrcholov dostaneme identický graf a $|\mathrm{iso}(C_3)| = 1$. Na druhej strane, P_3 má dva automorfizmy (vrátane identity) a C_3 ich má 6 (všetky permutácie). Definujme $\mathrm{aut}(G) = \{\pi \mid \pi(G) = G\}$. Potom platí:

$$|\mathrm{iso}(G)| \cdot |\mathrm{aut}(G)| = n! = \text{veľkosť } \textit{multimnožiny } \{\pi(G) \mid \pi \text{ je permutácia}\}.$$

Namiesto pôvodnej voľby teda uvažujme

$$S = \{(H, \pi) \mid H \sim G_1 \vee H \sim G_2, \pi \in \mathrm{aut}(H)\}.$$

O tejto množine naozaj platí, že

$$|S| = \begin{cases} n! & \text{ak } G_1 \sim G_2 \\ 2n! & \text{ak } G_1 \not\sim G_2 \end{cases}$$

bez ohľadu na automorfizmy G_1 a G_2.

Ukážeme všeobecný protokol pre ľubovoľnú množinu S taký, že ak $|S| \geq K$, dokazovateľ to vie dokázať s vysokou pravdepodobnosťou; naopak, ak $|S| \leq K/2$, overovateľ s vysokou pravdepodobnosťou odmietne.

Nech $S \subseteq \{0,1\}^m$, pričom príslušnosť do S sa dá certifikovať a obaja účastníci vedia hodnotu K. Nech $k \in \mathbb{N}$ je také, že $2^{k-2} < K \leq 2^{k-1}$. Artur vyberie náhodnú hešovaciu funkciu $h : \{0,1\}^m \to \{0,1\}^k$ zo silne univerzálnej rodiny hešovacích funkcií $H_{m,k}$, zvolí náhodné $y \in_R \{0,1\}^k$ a pošle Merlinovi

dvojicu (h, y). Merlin skúsi nájsť $x \in S$ také, že $h(x) = y$ a pošle Arturovi hodnotu x a certifikát, že $x \in S$. Merlin overí certifikát a to, že $h(x) = y$ a ak všetko sedí, akceptuje. Rozdiel vo veľkosti $|S|$ sa odrazí v rozdielnej pravdepodobnosti akceptácie.

Konkrétne, nech $p^* = K/2^k$ a nech $A = \Pr_{h,y}[\exists x : h(x) = y]$ je pravdepodobnosť akceptovania. Ukážeme, že

$$\begin{aligned} A &\leq \frac{1}{2}p^* \qquad \text{ak } |S| < K/2 \\ A &\geq \frac{3}{4}p^* \qquad \text{ak } |S| \geq K \end{aligned}$$

V prvom prípade je pravdepodobnosť zjavne $\leq |S|/2^k \leq p^*/2$. V druhom prípade nech E_x je udalosť, že $h(x) = y$ pre $x \in S$. Z princípu inklúzie-exklúzie je

$$\Pr_h[\exists x : h(x) = y] = \Pr[\bigvee_{x \in S} E_x] \geq \sum_{x \in S} \Pr[E_x] - \frac{1}{2} \sum_{x \neq x' \in S} \Pr[E_x \wedge E_{x'}].$$

$\Pr[E_x] = 2^{-k}$ a, vďaka univerzálnosti $H_{m,k}$, $\Pr[E_x \wedge E_{x'}] = 2^{-2k}$. Teda

$$A \geq \frac{|S|}{2^k} - \frac{1}{2}\frac{|S|^2}{2^{2k}} \geq \frac{|S|}{2^k}\left(1 - \frac{|S|}{2^{k+1}}\right) \geq p^*\left(1 - \frac{2^{k-1}}{2^{k+1}}\right) = \frac{3}{4}p^*$$

Merlin si spočíta $p^* = K/2^k$, pošle dokazovateľovi niekoľko náhodných dvojíc h, y a akceptuje, ak je podiel akceptovaných dvojíc aspoň $5p^*/8$. □

20.3 Triedy AM a IP

Definícia 20.1. *Nech $f, g : \{0,1\}^* \to \{0,1\}^*$, $k \in \mathbb{N}$; potom k-kolovú interakciu f a g na vstupe x (značíme $\langle f, g\rangle(x)$) definujeme ako postupnosť $a_1, a_2, \ldots, a_k \in \{0,1\}^*$ takú, že $a_1 = f(x)$, $a_2 = g(x, a_1)$, ..., $a_{2i+1} = f(x, a_1, \ldots, a_{2i})$, $a_{2i+2} = g(x, a_1, \ldots, a_{2i+1})$. Výstup na konci interakcie je $\mathsf{out}_f\langle f, g\rangle(x) = f(x, a_1, \ldots, a_k) \in \{0,1\}$.*

Ak chceme definovať interakciu s pravdepodobnostným overovateľom, vstupom je ešte reťazec náhodných bitov r a definujeme $a_1 = f(x, r)$, $a_2 = g(x, a_1)$, $a_3 = f(x, r, a_1, a_2)$, atď. V tomto prípade funkcia g „nevidí“ náhodný reťazec a dostávame model so súkromnou mincou. Naopak, môžeme definovať $a_2 = g(x, r, a_1)$, atď. model s verejnou mincou. V oboch prípadoch je $\mathsf{out}_f\langle f, g\rangle(x)$ náhodná premenná – výsledok závisí od náhodného reťazca r.

Definícia 20.2 (IP). *$L \in \mathsf{IP}[k]$ (kde k môže závisieť od dĺžky vstupu), ak existuje PTS V (overovateľ) bežiaci v polynomiálnom čase taký, že pri k-kolovej interakcii s funkciou P (dokazovateľ)*

- *$x \in L \Rightarrow \exists P : \Pr_r[\mathsf{out}_V\langle V, P\rangle(x) = 1] \geq 2/3$ (úplnosť)*

- $x \notin L \Rightarrow \forall P : \Pr_r[\mathsf{out}_V\langle V, P\rangle(x) = 1] \leq 1/3$ *(korektnosť).*

Definujeme $\mathsf{IP} = \bigcup_c \mathsf{IP}[n^c]$.

- konštanty $2/3$ a $1/3$ sa dajú nahradiť $1 - 1/2^{n^s}$ a $1/2^{n^s}$ pre ľubovoľnú fixnú konštantu s.
- pre daného overovateľa V vieme vypočítať optimálneho dokazovateľa P v polynomiálnom čase, teda môžeme predpokladať, že $P \in \mathsf{PSPACE}$ a $\mathsf{IP} \subseteq \mathsf{PSPACE}$.
- ak v definícii konštantu $2/3$ nahradíme 1, trieda IP sa nezmení (netriviálne)
- naopak, ak v definícii konštantu $1/3$ nahradíme 0, dostaneme len NP

Definícia 20.3 (AM)**.** *Pre každé k definujme* $\mathsf{AM}[k]$ *ako podmnožinu* $\mathsf{IP}[k]$*, ktorú dostaneme, ak obmedzíme overovateľa nasledovne: správy, ktoré posiela sú náhodné bity, ktoré má k dispozícii a pri výpočte nemôže používať iné náhodné bity ako tie, čo poslal.*

Toto je verzia IP s verejnou mincou. Všimnite si, že dokazovateľ nedostane všetky náhodné bity naraz.

Špeciálne $\mathsf{AM} = \mathsf{AM}[2]$: Artur povie Merlinovi výsledky svojich hodov mincou; Merlin odpovie dôkazom pre tieto hody a Artur sa snaží skontrolovať dôkaz (ale už nehádže mincou).

Definícia 20.4. AM *je trieda jazykov, pre ktoré existuje deterministický TS V, bežiaci v polynomiálnom čase od x taký, že*

- *ak $x \in L$, tak* $\Pr_y[\exists z : V(x, y, z) = 1] \geq 2/3$
- *ak $x \notin L$, tak* $\Pr_y[\exists z : V(x, y, z) = 1] \leq 1/3$.

V MA najskôr hovorí Merlin; Artur si hodí mincou a snaží sa skontrolovať dôkaz.

Definícia 20.5. MA *je trieda jazykov, pre ktoré existuje deterministický TS V, bežiaci v polynomiálnom čase od x taký, že*

- *ak $x \in L$, tak* $\exists y \Pr_z[V(x, y, z) = 1] \geq 2/3$
- *ak $x \notin L$, tak* $\forall y : \Pr_z[V(x, y, z) = 1] \leq 1/3$.

Zjavne $\mathsf{BPP} \subseteq \mathsf{MA}$ (Artur odignoruje dôkaz a sám si vstup overí) a $\mathsf{NP} \subseteq \mathsf{MA}$ (Artur nepoužije náhodné hody mincou a overí dôkaz deterministicky). AM sa dá považovať za randomizovanú verziu NP. Sú to všetky problémy, ktoré sa dajú redukovať na Sat *randomizovanou* polynomiálnou redukciou.

Veta 20.3 (Goldwasser, Sipser). *Pre každé $k(n) : \mathbb{N} \to \mathbb{N}$, kde $k(n)$ je vypočítateľné v polynomiálnom čase, je* $\mathsf{IP}[k] \subseteq \mathsf{AM}[k + 2]$.

Dôkaz je trochu technický, my si dokážeme len špeciálny prípad: protokol pre grafového neizomorfizmus (veta 20.1) sa dá upraviť tak, aby používal iba verejnú mincu.

20.4 Perfektná úplnosť

Veta 20.4 (Perfektná úplnosť MA). *Každý* MA *protokol sa dá upraviť tak, aby v prípade, že $x \in L$ Artur* vždy *akceptoval. Teda konštanta $2/3$ v definícii 20.5 sa dá nahradiť 1.*

■ **Dôkaz.** Dôkaz je podobný ako pri $\mathsf{BPP} \subseteq \Sigma_2^\mathsf{P}$: Označme S_x množinu náhodných reťazcov, na ktorých Artur akceptuje. Opakovaním vieme dosiahnuť, že ak $x \in L$, $|S_x| \geq 1 - 1/2^n$, ale ak $x \notin L$, $|S_x| \leq 1/2^n$. Potom ale ak $x \in L$, existujú posunutia $t_0, \ldots, t_k$ také, že $\bigcup_{i=0}^{k} S \oplus t_i = \{0,1\}^m$; naopak, ak $x \notin L$, zjednotenie bude stále príliš malé. Merlin teda okrem svojho dôkazu pošle Arturovi aj posunutia $t_0, \ldots, t_k$; ten si nahádže mincou náhodný reťazec r a overí, či by aspoň na jednom $r \oplus t_i$ akceptoval. □

Veta 20.5. $\mathsf{MA} \subseteq \mathsf{AM}$. *Navyše v každom Artur-Merlinovom protokole vieme nahradiť dve po sebe idúce kolá Merlin-Artur poradím Artur-Merlin a ak bol pôvodný protokol perfektne úplný, aj nový bude.*

■ **Dôkaz.** Predpokladajme, že MA protokol má perfektnú úplnosť; na vstupe x Merlin pošle správu m a Artur ju pomocou náhodného reťazca r overí. AM protokol bude vyzerať nasledovne: Artur pošle Merlinovi náhodný reťazec r a Merlin mu odpovie m. Vďaka perfektnej úplnosti, ak $x \in L$, odpoveď m Artura presvedčí vždy. Treba ešte dokázať, že ak $x \notin L$, Merlin Artura nepresvedčí. Uvedomte si, že vďaka vymeneniu krokov teraz Merlin dopredu vie, s akým náhodným reťazcom r bude Artur kontrolovať jeho dôkaz. Avšak pre každé m' existuje len málo náhodných reťazcov, na ktorých Artur akceptuje (pravdepodobnosť stlačíme pod $1/2^{k+1}$, kde k je dĺžka Merlinovej správy). Korektnosť potom vyplýva z union boundu: pravdepodobnosť, že existuje m', ktorá Artura presvedčí $\leq 2^k$-krát pravdepodobnosť, že konkrétna m' Artura presvedčí a to je najviac polovica. □

Veta 20.6 (Perfektná úplnosť AM). *Každý* AM *protokol sa dá upraviť tak, aby v prípade, že $x \in L$ Artur* vždy *akceptoval. Teda konštanta $2/3$ v definícii 20.4 sa dá nahradiť 1.*

■ **Dôkaz.** Dôkaz je podobný ako pri MA; najskôr však spravíme MAM-protokol, ten vieme pomocou predchádzajúcej vety upraviť na AM-protokol: Merlin povie posunutia $t_0, \ldots, t_k$, Artur mu pošle náhodný reťazec r a Merlin mu dokáže, že $r \in \bigcup_{i=0}^{k} S \oplus t_i$: pošle mu index i a dôkaz m pre $r \oplus t_i$. Nakoniec Artur už len overí dôkaz m s náhodnými bitmi $r \oplus t_i$ tak, ako v pôvodnom protokole. Takto dostávame ekvivalentný MAM-protokol s perfektnou úplnosťou. Z predchádzajúcej vety vieme, že kroky „MA“ vieme vymeniť za „AM“ (pričom sa zachová perfektná úplnosť) a dve správy môže, samozrejme, Merlin poslať aj naraz. □

20.5 Dôsledky

Dôsledok 20.1 (Kolaps AM hierarchie). *Pre konštantné k je* $\mathsf{AM}[k] = \mathsf{IP}[k] = \mathsf{AM}[2] = \mathsf{AM}$.

■ **Dôkaz.** Postupne nahrádzame „MA“-kroky krokmi „AM“, napríklad pre $\mathsf{AM}[6]$ máme $\mathsf{AMAMAM} = \mathsf{AAMMAM} = \mathsf{AMAM} = \mathsf{AAMM} = \mathsf{AM}$. □

Dôsledok 20.2. $\mathsf{MA} \subseteq \Sigma_2^{\mathsf{P}} \cap \Pi_2^{\mathsf{P}}$, $\mathsf{AM} \subseteq \Pi_2^{\mathsf{P}}$.

■ **Dôkaz.** Vďaka perfektnej úplnosti vieme triedu MA charakterizovať takto:

- ak $x \in L$, $\exists m \forall r : V(x, m, r)$ akceptuje
- ak $x \notin L$, $\forall m$ pre väčšinu $r : V(x, m, r)$ odmietne

naproti tomu v Σ_2^{P} ak $x \notin L$, stačí, aby $\forall m \exists r : V(x, m, r)$ odmietne.

Podobne pre $L \in \mathsf{AM}$ platí:

- ak $x \in L$, $\forall r \exists m : V(x, m, r)$ akceptuje
- ak $x \notin L$, pre väčšinu r $\forall m : V(x, m, r)$ odmietne

naproti tomu v Π_2^{P} ak $x \notin L$, stačí, aby $\exists r \forall m : V(x, m, r)$ odmietne. □

Veta 20.7 (Boppana, Håstad, Zachos). *Ak* $\mathsf{coNP} \subseteq \mathsf{AM}$, *tak* $\mathsf{PH} = \Sigma_2^{\mathsf{P}}$.

■ **Dôkaz.** Zjavne $\mathsf{AM} \subseteq \Pi_2^{\mathsf{P}}$, takže stačí ukázať, že z predpokladu vyplýva $\Sigma_2^{\mathsf{P}} \subseteq \mathsf{AM}$. Nech $L \in \Sigma_2^{\mathsf{P}}$, potom existuje $L' \in \Pi_1^{\mathsf{P}} = \mathsf{coNP}$ taký, že $x \in L \iff \exists y : (x, y) \in L'$. Ale podľa predpokladu $L' \in \mathsf{AM}$; núka sa nám takýto protokol: Merlin povie y a potom spustíme AM protokol pre L'. Dostávame $L \in \mathsf{MAM} \subseteq \mathsf{AM}$, teda $\Sigma_2^{\mathsf{P}} \subseteq \mathsf{AM}$. □

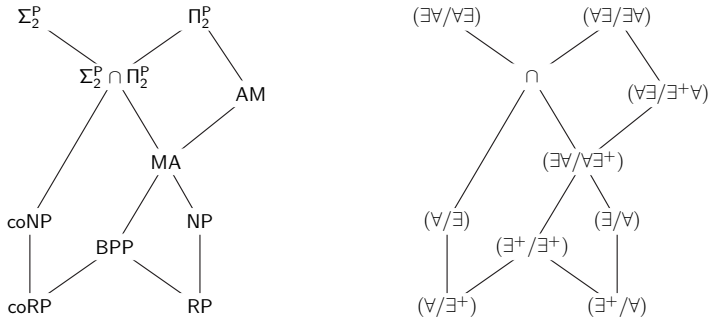
Ukázali sme si, že podľa Ladnerovej vety, ak $\mathsf{P} \neq \mathsf{NP}$, tak existujú problémy medzi P a NPC. Horúcim kandidátom na takýto problém je grafový izomorfizmus:

Dôsledok 20.3. *Ak grafový izomorfizmus je* NP*-úplný problém, tak* $\mathsf{PH} = \Sigma_2^{\mathsf{P}}$.

■ **Dôkaz.** Keby $\mathrm{GI} \in \mathsf{NPC}$, tak GNI je coNP-úplný, teda $\mathsf{coNP} \subseteq \mathsf{AM}$. □

20.6 Zhrnutie

Literatúra



Obr. 20.2: Vzťahy medzi triedami

Kapitola 21

Interaktívne protokoly

Veta 21.1 (Lund a spol. (1992)). $\mathsf{NP} \subseteq \mathsf{IP}$ *aj* $\mathsf{coNP} \subseteq \mathsf{IP}$. *Merlin vie dokonca Arturovi dokázať, aký je presný počet splňujúcich ohodnotení danej formuly,* $\mathsf{P}^{\#\mathsf{P}} \subseteq \mathsf{IP}$.

■ **Dôkaz.** Ukážme si protokol na konkrétnom príklade formuly

$$\phi \equiv (x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (x \vee \neg y \vee z).$$

Formula ϕ má presne 5 splňujúcich ohodnotení:

x	y	z	$\phi(x,y,z)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

K formule ϕ najskôr zostrojíme polynóm f taký, že ohodnotenie x, y, z splňuje (resp. nesplňuje) ϕ práve vtedy, keď $f(x,y,z) = 1$ (resp. 0). Pravdivostné hodnoty budeme reprezentovať ako prirodzené čísla: 0 je nepravda, 1 je pravda.

Aritmetizácia:

formula $\phi(x,y,z)$	$\rightarrow$	polynóm $f(x,y,z)$
false	$\rightarrow$	0
true	$\rightarrow$	1
$x \in \{\mathsf{false}, \mathsf{true}\}$	$\rightarrow$	$x \in \{0,1\}$
$\neg x$	$\rightarrow$	$1 - x$
$x \wedge y$	$\rightarrow$	$x \cdot y$
$x \vee y$	$\rightarrow$	$1 - (1-x)(1-y)$

$$\begin{aligned}\phi(x,y,z) &\equiv (x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (x \vee \neg y \vee z)\\ f(x,y,z) &= (1-(1-x)(1-y)(1-z))\\ &\quad \cdot (1-x(1-y)(1-z))\\ &\quad \cdot (1-(1-x)y(1-z))\end{aligned}$$

Fakt, že ϕ má 5 splňujúcich ohodnotení je ekvivalentná tvrdeniu, že

$$\sum_{x\in\{0,1\}} \sum_{y\in\{0,1\}} \sum_{z\in\{0,1\}} f(x,y,z) = 5.$$

Ako nás o tom môže Merlin presvedčiť?

Artur: [Zvolí si vhodne veľké prvočíslo (alebo si ho jednoducho vypýta od Merlina a overí).] Všetko budeme počítať modulo 13.

Artur: Uvažujme sumu $\sum_{y\in\{0,1\}} \sum_{z\in\{0,1\}} f(x,y,z)$. To znamená celý výraz okrem vonkajšej sumy. Toto je polynóm v x. Ako vyzerá tento polynóm?

Merlin: $g_1(x) = -x^2 + 2x + 2$.

Artur: [Ak hovorí Merlin pravdu, potom $\sum_{x\in\{0,1\}} \sum_{y\in\{0,1\}} \sum_{z\in\{0,1\}} f(x,y,z) = \sum_{x\in\{0,1\}} g_1(x) = g_1(0)+g_1(1)$. Dosadí teda 0 a 1 a overí, že $g_1(0)+g_1(1) = 2+3 = 5$.]

Artur: [A teraz príde výzva! Zvolí úplne náhodné číslo pre x (mod 13) a dosadí. Zvoľme napr. náhodne $x = 8$, $g_1(8) = 6$.] Ak hovoríš pravdu, potom $\sum_{y\in\{0,1\}} \sum_{z\in\{0,1\}} f(8,y,z) = 6$. Dokáž!

Ako? Protokol bude prebiehať podobne, opäť odstránime jednu vonkajšiu sumu:

Artur: Uvažujme sumu $\sum_{z\in\{0,1\}} f(8,y,z)$. To je polynóm v y. Ako vyzerá tento polynóm?

Merlin: $g_2(y) = 11y^3 + 7y^2 + 7y + 10$.

Artur: [Ak je to tak, potom $\sum_{y\in\{0,1\}} \sum_{z\in\{0,1\}} f(8,y,z) = \sum_{y\in\{0,1\}} g_2(y) = g_2(0) + g_2(1)$. Dosadí 0 a 1 a overí, že $g_2(0) + g_2(1) = 10 + 9 = 6$. To sedí.]

Artur: [Zvolí náhodné číslo pre y (mod 13) a dosadí do g_2. Zvoľme napr. náhodne $y = 5$, $g_2(5) = 9$.] Ak hovoríš pravdu, potom $\sum_{z\in\{0,1\}} f(8,5,z) = 9$. Dokáž!

Ako? Odstránime aj poslednú sumu:

Artur: Uvažujme $f(8,5,z)$ – polynóm v z. Ako vyzerá tento polynóm?

Merlin: $g_3(z) = 4z^3 + 12z^2 + 3z + 8$

Artur: [Dosadí 0 a 1 a overí, že $g_3(0) + g_3(1) = 8 + 1 = 9$.]

Artur: [Zvolí náhodné číslo pre z (mod 13) a dosadí. Zvoľme napr. náhodne $z = 11$.] Ak hovoríš pravdu, potom $f(8, 5, 11) = g_3(11) = 5$. Sedí to? Dosaďme $x = 8, y = 5, z = 11$ do polynómu f:

$$\begin{aligned} f(8,5,11) &= (1-(1-8)(1-5)(1-11)) \\ &\quad \cdot (1-8(1-5)(1-11)) \\ &\quad \cdot (1-(1-8)5(1-11)) = 5 \pmod{13} \end{aligned}$$

[Všetko v poriadku, Artur akceptuje.]

Dokáže nás Merlin presvedčiť o nepravde? Napríklad, mohol by nás Merlin presvedčiť, že $\sum_{x\in\{0,1\}} \sum_{y\in\{0,1\}} \sum_{z\in\{0,1\}} f(x, y, z) = 1$??

Uvažujme sumu $\sum_{y\in\{0,1\}} \sum_{z\in\{0,1\}} f(x, y, z)$. Ako sme už spomínali, toto je polynóm v x, konkrétne $g_1(x) = -x^2 + 2x + 2$, ale Merlin nemôže povedať $g_1(x)$, pretože $g_1(0) + g_1(1) = 5$ a nie 1 a Artur by to hneď odhalil. Merlin teda musí zaklamať a povedať nejaký iný polynóm, napríklad $h_1(x) = x^2 + 5x + 4$.

Artur dosadí 0 a 1 a overí, že $h_1(0) + h_1(1) = 10 + 4 = 1$. Zvolí náhodné x, napríklad 3 a dosadí $h_1(3) = 2$. Dokáže Merlin Artura presvedčiť, že $\sum_{y\in\{0,1\}} \sum_{z\in\{0,1\}} f(3, y, z) = 2$, hoci správna odpoveď je $g_1(3) = 12$? Uvažujme sumu $\sum_{z\in\{0,1\}} f(3, y, z)$, čo je polynóm v y, konkrétne $g_2(y) = y^3 + 7y^2 + y + 8$, ale Merlin nemôže povedať $g_2(y)$, lebo $g_2(0)+g_2(1) = 12$ a nie 2. Merlin teda musí opäť zaklamať a povedať nejaký iný polynóm, napríklad $h_2(y) = y^3 + 7y^2 + y + 3$.

Artur: dosadí 0 a 1 a overí, že $h_2(0) + h_2(1) = 12 + 3 = 2$. Zvolí náhodné y, napríklad 7 a dosadí $h_2(7) = 7$. Dokáže Merlin Artura presvedčiť, že $\sum_{z\in\{0,1\}} f(3, 7, z) = 7$, hoci správna odpoveď je $g_2(7) = 12$? Uvažujme $f(3, 7, z) = g_3(z) = 8z^3 + 8z + 11$, ale Merlin nemôže povedať $g_3(z)$, lebo $g_3(0) + g_3(1) = 12 \neq 7$. Merlin musí opäť klamať, napríklad, že $f(3, 7, z) = h_3(z) = z^3 + 2z + 2$.

Artur dosadí 0 a 1 a overí, že $h_3(0) + h_3(1) = 2 + 5 = 7$. Zvolí náhodné z, napríklad 2 a dosadí $h_3(2) = 1$. Keď však nakoniec vyhodnotí $f(3, 7, 2)$, zistí, že

$$\begin{aligned} f(3,7,2) &= (1-(1-3)(1-7)(1-2)) \\ &\quad \cdot (1-3(1-7)(1-2)) \\ &\quad \cdot (1-(1-3)7(1-2)) = 0 \neq 1 \pmod{13} \end{aligned}$$

a teda Merlin celý čas klamal.

Vo všeobecnosti pre daný polynóm $f(x_1, \ldots, x_n)$, celé číslo K a prvočíslo p chceme overiť, že

$$\sum_{x_1\in\{0,1\}} \sum_{x_2\in\{0,1\}} \cdots \sum_{x_n\in\{0,1\}} f(x_1, \ldots, x_n) = K.$$

Artur: Ak $n = 0$, overí, že $f = K$. Ak áno, akceptuje, inak odmietne. Pre $n \geq 1$ sa spýta Merlina na polynóm $\sum_{x_2\in\{0,1\}} \cdots \sum_{x_n\in\{0,1\}} f(x_1, \ldots, x_n) = g(x_1)$.

Merlin: Odpovie nejakým polynómom $h(x_1)$ (ak nepodvádza, mal by odpovedať $g(x_1)$).

Artur: Overí si, že $h(0) + h(1) = K$ (ak nie, ihneď odmietne). Zvolí si náhodné číslo $a \pmod p$ a rekurzívne overí, že

$$\sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} f(a, x_2 \ldots, x_n) = h(a).$$

Aká je šanca, že $h(a) = g(a)$ pre náhodné $a \pmod p$? Pracujeme s polynómami stupňa $\leq n$ a $h(a) = g(a)$ práve vtedy, keď a je koreň polynómu $h - g$, čo je tiež polynóm stupňa $\leq n$. A teda pravdepodobnosť, že Merlin zrovna trafí koreň je $\leq n/p$.

Naopak s pravdepodobnosťou $(1 - n/p)$ bude $h(a) \neq g(a)$. Indukciou sa dokáže, že

Ak je hodnota nepravdivá, dôkaz odmietneme s pravdepodobnosťou aspoň

$$\left(1 - \frac{n}{p}\right)\left(1 - \frac{n}{p}\right) \cdots \left(1 - \frac{n}{p}\right) = \left(1 - \frac{n}{p}\right)^n \approx 1 - \frac{n^2}{p}$$

Ak teda na začiatku zvolíme dostatočne veľké p medzi 2^n a 2^{n+1}, pravdepodobnosť oklamania je zanedbateľná. □

Veta 21.2 (Shamir (1992), Shen (1992)). IP = PSPACE.

■ **Dôkaz.** Stačí ukázať interaktívny protokol pre QBF. Mohli by sme skúsiť pokračovať v aritmetizácii:

formula $\phi(x, y, z)$	$\rightarrow$	polynóm $f(x, y, z)$
$(\forall x)$	$\rightarrow$	$\prod_{x \in \{0,1\}}$
$(\exists x)$	$\rightarrow$	$\coprod_{x \in \{0,1\}}$,
	$\rightarrow$	kde $\coprod_x f(x) = 1 - \prod_x (1 - f(x))$

Napríklad formula

$$(\forall x)(\exists z)(\forall y)(x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (x \vee \neg y \vee z)$$

je pravdivá práve vtedy, keď

$$\prod_{x \in \{0,1\}} \coprod_{z \in \{0,1\}} \prod_{y \in \{0,1\}} f(x, y, z) = 1.$$

Ak by sme však chceli použiť vyššie uvedený protokol, problém sa ukáže už v prvom kroku. Predstavme si, že začneme s výrazom $\prod_{z \in \{0,1\}} \prod_{x_1=0}^{1} \prod_{x_2=0}^{1} \cdots \prod_{x_n=0}^{1} z$. Artur odstráni vonkajší súčin a spýta sa na $\prod_{x_1=0}^{1} \prod_{x_2=0}^{1} \cdots \prod_{x_n=0}^{1} z = z^{2^n}$. Toto je polynóm v z – avšak exponenciálne veľkého stupňa!

Ako tento problém zaplátame? Výsledok polynómu nás zaujíma iba pre hodnoty 0 a 1 a pre $x \in \{0,1\}$ platí $x^k = x$.

Pridajme teda linearizačný operátor L_x, ktorý definujeme nasledovne:

$$L_x f(x) = (1-x)f(0) + xf(1) = f(0) + (f(1) - f(0))x.$$

Jednoducho bodmi 0 a 1 preložíme priamku. Všimnite si, $f(x)$ a $L_x f(x)$ dávajú rovnaké hodnoty pre $x \in \{0,1\}$. Čo je však podstatné, nech je $f(x)$ polynóm ľubovoľne veľkého stupňa, $L_x f(x)$ je už iba lineárny.

Ekvivalentne $L_x f(x) = f(x) \bmod x^2 - x$. Prečo? Vypočítať zvyšok po delení $x^2 - x$ je to isté ako povedať, že $x^2 - x \equiv 0$, čo je to isté, ako povedať, že $x^2 \equiv x$, čo platí pre $x = 0$ a $x = 1$. Navyše zvyšok po delení je polynóm menšieho stupňa, v tomto prípade lineárny.

Formula

$$(\forall x)(\exists z)(\forall y)(x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (x \vee \neg y \vee z)$$

je pravdivá práve vtedy, keď

$$\prod_{x\in\{0,1\}} L_x \coprod_{z\in\{0,1\}} L_x L_z \prod_{y\in\{0,1\}} L_x L_y L_z f(x,y,z) = 1$$

a vďaka linearizačnému operátoru budú všetky polynómy malého stupňa.

Ak teda chceme dokázať, že

$$\mathcal{O}_1 \mathcal{O}_2 \cdots \mathcal{O}_\ell f(x_1, \ldots, x_n) = K,$$

kde $\mathcal{O}_i \in \{\prod, \coprod, L\}$ sú rôzne operátory.

Artur: Ak $n = 0$, overí, že $f = K$. Ak áno, akceptuje, inak odmietne. Pre $n \geq 1$ (premenujme premenné tak, že $\mathcal{O}_1$ operuje na x_1) sa spýta Merlina na polynóm $\mathcal{O}_2 \cdots \mathcal{O}_\ell f(x_1, \ldots, x_n) = g(x_1)$.

Merlin: Pošle $h(x_1)$ (ak nepodvádza, mal by odpovedať $g(x_1)$).

Artur: Skontroluje:

- pre $\mathcal{O}_1 = \prod$, či $h(0) \cdot h(1) = K$,
- pre $\mathcal{O}_1 = \coprod$, či $1 - (1 - h(0))(1 - h(1)) = K$,
- pre $\mathcal{O}_1 = L_{x_1}$, či $(L_{x_1} h)(a_1) = a_1 h(0) + (1 - a_1)h(1) = K$.

Ak nie, ihneď odmietne. Zvolí si náhodné číslo $a \pmod p$ a rekurzívne overí, že

$$\mathcal{O}_{x_2} \cdots \mathcal{O}_{x_n} f(a, x_2 \ldots, x_n) = h(a).$$

□

Literatúra

Lund, Carsten, Lance Fortnow, Howard Karloff, a Noam Nisan. 1992. "Algebraic Methods for Interactive Proof Systems." *Journal of the ACM (JACM)* 39(4), s. 859–868.

Shamir, Adi. 1992. "IP = PSPACE." *Journal of the ACM (JACM)* 39(4), s. 869–877.

Shen, Alexander. 1992. "IP = PSPACE: simplified proof." *Journal of the ACM (JACM)* 39(4), s. 878–880.

Kapitola 22

Interaktívne dôkazy s viacerými dokazovateľmi

22.1 MIP

Definícia 22.1 (MIP). *Model: jeden PTS V (overovateľ) bežiaci v polynomiálnom čase a k neobmedzených dokazovateľov $P_1, \ldots, P_k$. V s každým dokazovateľom zdieľa osobitnú komunikačnú pásku; všetci účastníci zdieľajú read-only vstupnú pásku, avšak žiadni dvaja dokazovatelia okrem toho nemajú nič spoločné a nedokážu spolu komunikovať. $L \in k$-MIP, ak existuje PTS V (overovateľ) bežiaci v polynomiálnom čase taký, že*

- $x \in L \Rightarrow \exists P_1, \ldots, P_k : \Pr_r[\mathsf{out}_V\langle V, P_1, \ldots, P_k\rangle(x) = 1] \geq 2/3$ *(úplnosť)*
- $x \notin L \Rightarrow \forall P_1, \ldots, P_k : \Pr_r[\mathsf{out}_V\langle V, P_1, \ldots, P_k\rangle(x) = 1] \leq 1/3$ *(korektnosť).*

Definujeme $\mathsf{MIP} = \bigcup_k k\text{-MIP}$. *(MIP je skratka pre multi-prover interactive protocol).*

Veta 22.1. $\mathsf{PCP}[\mathsf{poly}, \mathsf{poly}] = \mathsf{MIP} = 2\text{-MIP}$.

■ **Dôkaz.** Simulácia PCP M dvomi dokazovateľmi: V sa spýta prvého dokazovateľa všetky na všetky miesta v dôkaze, ktoré sa pýta M. Potom si vyberie jednu náhodnú z položených otázok a spýta sa ju aj druhého dokazovateľa a overí, že odpovede boli rovnaké. (Rozdiel medzi dôkazom a otázkami pre P_1 je, že interaktívny dokazovateľ sa môže rozhodnúť, ako odpovedať na ďalšie otázky podľa tých predošlých; naproti tomu P_2 históriu nepozná.)

Zjavne, ak $x \in L$, existuje dôkaz π, ktorý vie M overiť a ak budú P_1 aj P_2 odpovedať podľa π, presvedčia V. Naopak, ak $x \notin L$, predstavme si, že odpovede P_2 na všetky otázky dopredu napíšeme ako reťazec π (aj tak sa P_2 pýtame iba raz). Ak P_1 odpovie rovnako, ako π, tak každý dôkaz presvedčí M s pravdepodobnosťou najviac $1/3$ a ak P_1 hovorí niečo iné, V to odhalí s

pravdepodobnosťou aspoň $1/n^k$ (kde n^k je počet otázok). Opakovaním protokolu sa táto pravdepodobnosť dá zmenšiť pod 1/3.

Naopak, aj viacerých interaktívnych dokazovateľov vieme zakódovať ako dôkaz: jeho $(i, j, k, \beta_{i1}, \ldots, \beta_{ij})$-ty bit bude k-ty bit j-tej odpovede dokazovateľa P_i na otázku β_{ij}, pričom $\beta_{i1}, \ldots, \beta_{i(j-1)}$ sú všetky predošlé otázky, na ktoré odpovedal. □

$f \approx_\delta g$, ak $\Pr_x[f(x) \neq g(n)] \leq \delta$

Ak $f_1 \approx_\delta g$ aj $f_2 \approx_\delta g$ a f_1, f_2 sú polynómy malého stupňa, potom aj $f = f_1 - f_2$ je malého stupňa, ktorý je nulový na 2δ vstupov z I^m, takže ak $|I| \geq 2dm$ a $\delta \leq 1/4$, potom $f = 0$, teda $f_1 = f_2$ je jedinečný polynóm. Polynómy malého stupňa tvoria samoopravný kód.

Lema 22.1. *Nech π je dôkaz ktorý kóduje hodnoty nejakej funkcie h, t.j. na pozícii $(r_1, \ldots, r_m)$, kde $r_i \in I \subseteq \mathbb{F}$ je hodnota $h(r_1, \ldots, r_m) \in \mathbb{F}$. Nech $|I| = kdm$. Potom existuje polo-interaktívny protokol taký, že*

- *ak π kóduje polynóm h stupňa $\leq d$ v každej premennej a $\sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_m \in \{0,1\}} h(x_1, \ldots$ a, tak existuje dokazovateľ, ktorý overovateľa určite presvedčí,*
- *ak π kóduje funkciu, ktorá je δ-blízko ($\delta \leq 1/4$) polynómu f stupňa $\leq d$ v každej premennej a ak existuje dokazovateľ, ktorý presvedčí s pravdepodobnosťou väčšou ako $\delta + 1/k$, potom $\sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_m \in \{0,1\}} f(x_1, \ldots, x_m) = a$.*

■ **Dôkaz.** Protokol je úplne rovnaký ako v dôkaze vety TODO $\mathsf{NP} \subseteq \mathsf{IP}$ iba s tým, že úplne na konci polynóm h nevyhodnotíme, ale pozrieme sa na hodnotu v dôkaze π. Prvý bod je jasný. V druhom predpokladajme, že v dôkaze je zakódované f namiesto h (kde f je jedinečný polynóm malého stupňa, ktorý je δ-blízko h). Rovnako ako v dôkaze TODO dokazovateľ musí klamať v každom kole s pravdepodobnosťou $1 - 1/k$. Na konci je jeho jediná záchrana, že f a h sa líšia na v náhodnom bode $(r_1, \ldots, r_m)$, čo je s pravdepodobnosťou najviac δ. □

Definícia 22.2 (Oracle-3-SAT). *Nech z a b_1, b_2, b_3 sú reťazce premenných, $|z| = r$, $|b_i| = s$, označme $b = b_1 b_2 b_3$ a $w = zb$. Nech $t = \tau_1 \tau_2 \tau_3$. Nech $B(w, t)$ je booleovská formula s $r + 3s + 3$ premennými. Hovoríme, že funkcia $A : \{0,1\}^s \to \{0,1\}$ je splňujúce orákulum pre B, ak pre každé w je $B(w, A(b_1), A(b_2), A(b_3))$ splnené.*

Veta 22.2. ORACLE-3-SAT *je* NEXP*-úplný.*

■ **Dôkaz.** Pozrime sa na NEXP výpočet stroja M na vstupe x a vytvorme preň 3-CNF formulu Ψ ako v Cook-Levinovej vete (hoci v tomto prípade exponenciálne dlhú). Nech počet premenných je 2^s, kde $s = \mathsf{poly}(n)$. Označme premenné $X(b)$, kde $b \in \{0,1\}^s$. Každú klauzulu môžeme zapísať v tvare

$$C(b, f, X) = (\phi_1 \oplus X(b_1)) \vee (\phi_2 \oplus X(b_2)) \vee (\phi_3 \oplus X(b_3)),$$

kde $\phi_i = 1$, ak je premenná v klauzule negovaná, takže f kóduje znamienka premenných a b kóduje indexy premenných v klauzule. Nech $B_1(f,t) = (\phi_1 \oplus \tau_1) \vee (\phi_2 \oplus \tau_2) \vee (\phi_3 \oplus \tau_3)$;

Samotné klauzuly sú rozpoznateľné v polynomiálnom čase – existuje polynomiálny algoritmus, ktorý na vstupe x, b, f povie, či sa klauzula $C(b,f,X)$ nachádza v Ψ, alebo nie. Na tento výpočet môžeme opäť použiť Cook-Levinovu vetu a dostaneme formulu $B_2(u,f,b)$.

Preto $x \in L$ práve vtedy, keď existuje funkcia A taká, že A spĺňa $C(b,f,X)$, alebo $C(b,f,X)$ nie je klauzula Ψ: Teda

$$\Psi'(u,b,f,t) = B_1(f,t) \vee \neg B_2(u,b,f),$$

čo je úsporný (polynomiálne veľký) zápis formule Ψ. □

Lema 22.2. *Pre inštanciu* (B,r,s) Oracle-3-Sat *vieme v polynomiálnom čase vypočítať polynóm g s celočíselnými koeficientmi na rovnakej množine premenných taký, že $A : \{0,1\}^s \to Q$ je splňujúce orákulum pre B práve vtedy, keď* $\sum_{w \in \{0,1\}^{r+3s}} g(w, A(b_1), A(b_2), A(b_3)) = 0$.

■ **Dôkaz.** Tak ako pri IP, dostaneme polynóm f. Výledok je $g(w,t) = f(w,t)^2 + (\tau_1(\tau_1 - 1))^2$. Súčet štvorcov je nulový, ak sú všetky členy nulové, členy $(\tau_1(\tau_1 - 1))^2$ zaisťujú, že A nadobúda iba hodnoty $\{0,1\}$. □

MEH

Lema 22.3. *Nech $L \in$ NEXP. Potom existuje konštanta c taká, že pre všetky $x \in \{0,1\}^n$ existuje 3-CNF formula*

$$\Phi_x(X(0), X(1), \ldots, X(2^{n^c} - 1)) = \bigwedge_{i=0}^{2^{n^c}-1} C_i$$

exponenciálnej dĺžky s exponenciálne veľa (2^{n^c}) premennými, pričom každá klauzula C_i sa dá vypočítať v polynomiálnom čase z x a i a $x \in L$ práve vtedy, keď existuje ohodnotenie premenných $A : \{0,1\}^{n^c} \to \{0,1\}$, ktoré spĺňa všetky klauzuly.

Lema 22.4. *Nech $L \in$ NEXP. Potom existuje konštanta c a predikát $p_x : \{0,1\}^{4n^c+3} \to \{0,1\}$ vypočítateľný v polynomiálnom čase taký, že*

$$x \in L \iff \exists A : \forall i,b,t : \Big(p_x(i,b,t) \Rightarrow A(b_1), A(b_2), A(b_3) \text{ spĺňajú klauzulu } c[t]\Big)$$

$$\iff \exists A : \neg\exists i,b,t : \Big(p_x(i,b,t) \wedge \neg A(b_1), A(b_2), A(b_3) \text{ spĺňajú klauzulu } c[t]\Big),$$

kde $i \in \{0,1\}^{n^c}$ je index klauzuly, $b = b_1b_2b_3 \in \{0,1\}^{3n^c}$ sú čísla premenných a t je typ klauzuly. $p_x(i,b,t)$, ak i-ta klauzula C_i má tvar $t_1X(b_1) \vee t_2X(b_2) \vee t_3X(b_3)$.

Lema 22.5. *Existuje konštanta c' taká, že pre každé x existuje aritmetický výraz P_x vypočítateľný v polynomiálnom čase s $4n^c + n^{c'} + 6$ premennými stupňa najviac $n^{c'}$ taký, že pre každé $i, b_1, b_2, b_3 \in \{0,1\}^{n^c}$ a pre každé $t \in \{0,1\}^3$:*

$$\sum_{z \in \{0,1\}^{n^{c'}}} P_x(i, b, t, z, A(b_1), A(b_2), A(b_3))$$

je 1, ak $p_x(i, b, t) = 1 \wedge \neg A(b_1), A(b_2), A(b_3)$ spĺňajú klauzulu $c[t]$, inak 0.

■ **Dôkaz.** Z Cook-Levinovej vety máme F_x takú, že

$$\sum_z F_x(i, t, b, z) = [p_x(i, t, b)]$$

(z sú pomocné premenné); nech $q(t, a_1, a_2, a_3)$ je polynóm, ktorý aritmetizuje „a_1, a_2, a_3 spĺňajú klauzulu $c[t]$“. Potom $P_x(i, b, t, z, a) = F_x(i, t, b, z)(1 - q(t, a))$. □

Odteraz ďalej true bude 0 a false > 0.

$$\sum_{i,b,t,z} P_x(i, b, t, z, A(b_1), A(b_2), A(b_3)) = 0, \quad \text{ak neexistujú } i, b, t...$$

Lema 22.6. *Nech $L \in \mathsf{NEXP}$. Potom existujú konštanty c, c' taká, že pre každé x existuje aritmetický výraz P_x vypočítateľný v polynomiálnom čase s $4n^c + n^{c'} + 6$ premennými stupňa najviac $n^{c'}$ taký, že*

$$x \in L \iff \exists A : \sum_{i,b,t,z} P_x(i, b, t, z, A(b_1), A(b_2), A(b_3)) = 0,$$

kde $A : \{0,1\}^{n^c} \to \{0,1\}$ a $ibtz \in \{0,1\}^{n^{4c+c'+3}}$.

Namiesto A budeme používať A' – multilineárne rozšírenie. To, že $A'(b) \in \{0,1\}$ pre všetky $b \in \{0,1\}^{n^c}$ môžeme overiť tak, že overíme, že $\sum_{b \in \{0,1\}^{n^c}} [A'(b)(1 - A'(b))]^2 = 0$. Overí sa podobne ako v LFKN protokole, ale s použitím celých čísiel namiesto modulárnej aritmetiky. Dá sa dokázať, že koeficienty majú polynomiálne dlhý zápis. Nech $N = 9(n^{c'} + 3)(4n^c + n^{c'} + 3)$, všetky koeficienty sú najviac $2^{9n^c} N^{4n^c}$

Podobne sa otestuje (za predpokladu, že A' je naozaj multilineárne rozšírenie), či $\sum P_x = 0$

Existuje test multilineárnosti: pre $N \in \mathbb{N}$, $40n^2 < N \leq 2^n$ nech $I = \{0, 1, \ldots, N-1\}$. Nech $A : I^n \to Q$ je ľubovoľná funkcia. Existuje pravdepodobnostný test, ktorý v polynomiálnom čase overí, či je A multilineárna funkcia, pričom ak je, tak akceptuje a ak nie je aspoň 1/36-blízko multilineárnej funkcie g, tak akceptuje s pravdepodobnosťou najviac 1/9. Pri teste sa spýtame len na 4 hodnoty A' a pravdepodobnosť, že sa práve na nich A' líši od g je najviac $4/36 = 1/9$ – a ak sa zhodujú, overovateľ by akceptoval aj s dôkazom g namiesto

A', ale pre multilineárnu funkciu už vieme, že overovateľ by akceptoval s pravdepodobnosťou najviac 1/9. Takže overovateľ môže (mylne) akceptovať, ak a) A' je ďaleko od multilineárnej, ale nezistí to, b) A' je blízko multilineárnej funkcie g, ale zrovna na otázkach, ktoré sa overovateľ pýta, sa A' a g líšia, c) A' je blízko g, tie sa zhodujú na kontrolovaných hodnotách a dokazovateľovi sa predsa podarí overovateľa oklamať. Každá z týchto možností sa stane s pravdepodobnosťou najviac 1/9, spolu je to najviac 1/3.

22.2 Multilineárne rozšírenia

	0	1
0	1	2
1	1	4

	0	1	2	3	4
0	1	2			
1	1	4			
2					
3					
4					

	0	1	2	3	4
0	1	2	3	4	5
1	1	4	2	0	3
2	1	1	1	1	1
3	1	3	0	2	4
4	1	0	4	3	2

	0	1	2	3	4
0	1	2	3	4	5
1	1	3	0	2	4
2	1	4	2	0	3
3	1	0	4	3	2
4	1	1	1	1	1

$f(x,y) = a_0 + a_1 x + a_2 y + a_3 xy$ $1\cdot(1-x)(1-y) + 2\cdot(1-x)y + 1\cdot x(1-y) + 4\cdot xy = 2xy + y + 1$

Langrangeov interpolačný polynóm ako pri CNF pre

Vo všeobecnosti $\tilde{f}(x_1,\ldots,x_n) = \sum_{w\in\{0,1\}^n} f(w)\cdot\chi_w(x_1,\ldots,x_n)$, kde $\chi_w(x_1,\ldots,x_n) = \prod_{i=1}^{n}(w_i x_i + (1-w_i)(1-x_i))$. $\chi_w(w) = 1$ a $\chi_w(z) = 0$ pre všetky $z \neq w$, a teda $\sum f(w)\chi_w(z) = f(z)$ pre všetky $z \in \{0,1\}^n$.

Jedinečnosť: Ak p, q sú multilineárne a zhodujú sa na $\{0,1\}^n$, potom $p - q$ je tiež multilineárny a je nulový na $\{0,1\}^n$. Ukážeme, že je nulový na celom $\mathbb{F}^n$. Ak by to nebol nulový polynóm, nech t je člen najmenšieho stupňa. Ak dosadíme sa všetky premenné v t 1 a za všetky ostatné 0, člen t bude nenulový, ale všetky ostatné členy sú vyššieho alebo rovnakého stupňa a teda obsahujú nejakú premennú, ktorá nie je v t a ich hodnota bude nula. Takže sme ukázali, že nenulový polynóm nadobúda nenulovú hodnotu na $\{0,1\}^n$.

Literatúra

Časť VIII

Viac

Kapitola 23

„Zjemnená“ teória zložitosti

Uvažujme problém 3-SUM: dané sú množiny čísiel A, B, C ($|A| = |B| = |C| = n$) a cieľová suma t. Úlohou je nájsť $a \in A$, $b \in B$, $c \in C$ také, že $a + b + c = t$. Môžeme vyskúšať všetky trojice – riešenie v $O(n^3)$. Lepšie je vytvoriť si hešovaciu tabuľku, kde pre každý prvok $a \in A$ vložíme hodnotu $t - a$. Potom stačí skúšať už len všetky dvojice (b, c) a skontrolovať, či sa ich súčet nachádza v tabuľke – riešenie v $O(n^2)$. Alternatívne môžeme utriediť B a C a potom pre každé $a \in A$ hľadať dvojicu (b, c), ktorej súčet je $t - c$; toto hľadanie robíme inteligentne: prechádzame prvky B od najmenšieho k najväčšiemu a zároveň prvky C od najväčšieho k najmenšiemu.

zlepšené o $\sim 1/\log^2 n$; dolný odhad $\Omega(n^2)$ v modeli, kde otázky sú $\operatorname{sgn}(c_1 a + c_2 b + c_3 c)$.

Hypotéza 23.1. *3-SUM sa nedá riešiť v čase $O(n^{2-\varepsilon})$ pre $\varepsilon > 0$.*

k-SUM: Daná je cieľová suma t a k množín $A_1, \ldots, A_k$, každá veľkosti $|A_i| = n$. Vyber z každej množiny jeden prvok tak, aby súčet bol t.

k-SUM sa dá riešiť v čase $O(n^{\lceil k/2 \rceil})$.

k-SUM hypotéza: Neexistuje algoritmus, ktorý rieši k-SUM v čase $O(n^{\lceil k/2 \rceil - \varepsilon})$, tzn. existujúci algoritmus je v podstate optimálny.

Nech $c_k = \inf_c$k$-\textsc{Sum} \in \mathsf{DTIME}(n^c)$ je exponent v časovej zložitosti najrýchlejšieho algoritmu (resp. infimum, ak by existovala nekonečná postupnosť stále lepších algoritmov).

Veta 23.1. *Z ETH vyplýva, že $c_k = \Theta(k)$.*

Hypotéza 23.2 (ETH). *Hypotéza o exponenciálnom čase (Exponential Time Hypothesis): Pre každé $k \geq 3$ existuje $\varepsilon > 0$ také, že k-SAT sa nedá riešiť v čase $O(2^{\varepsilon n})$.*

Hypotéza 23.3 (SETH). *Silná hypotéza o exponenciálnom čase (Strong Exponential Time Hypothesis): Pre každé $\varepsilon > 0$ existuje k také, že k-SAT potrebuje čase $\Omega(2^{(1-\varepsilon)n})$.*

Inými slovami, nech s_k je najmenší koeficient δ taký, že k-SAT sa dá riešiť v čase $O(2^{\delta n}\mathsf{poly}(n))$ ($s_k = \inf\{\delta \mid \exists$alg. riešiaci k-SAT v čase $\tilde{O}(2^{\delta n})\}$) a nech $s_\infty = \lim_{k\to\infty} s_k$. ETH vraví, že $s_k > 0$ pre všetky $k \geq 3$. SETH vraví, že $s_\infty = 1$.

Problém kolmých vektorov – Ov (orthogonal vectors): Pre dané množiny d-rozmerných vektorov $A, B \subseteq \{0,1\}^d$ existuje dvojica kolmých vektorov $u \in A$, $v \in B$?

Hypotéza 23.4 (OVC). *Problém kolmých vektorov* Ov *pre* $d = \Omega(\log n)$ *sa nedá riešiť v čase* $O(n^{2-\varepsilon})$.

Veta 23.2. *SETH implikuje OVC.*

■ **Dôkaz.** Majme k-CNF formulu ϕ s premennými $x_1, \ldots, x_n$ a klauzulami $C_1, \ldots, C_m$ ($m = O(n)$). uvažujme všetky čiastnočné ohodnotenia α prvej polovice premenných (takých je $2^{n/2}$) a zostrojme vektory u^α také, že $u^\alpha_i = 0$, ak α spĺňa C_i, inak $u^\alpha_i = 1$. Podobne pre všetky čiastnočné ohodnotenia β druhej polovice premenných zostrojme vektory v^β, kde $v^\beta_i = 0$, ak β spĺňa C_i, inak $v^\beta_i = 1$. Potom α, β (ohodnotenie všetkých premenných) spĺňa ϕ práve vtedy, keď pre každé i aspoň jeden literál spĺňa C_i, a teda aspoň jedno z u^α_i, v^β_i je nula, čo je práve vtedy, keď $u^\alpha \perp v^\beta$. Formula ϕ je splniteľná práve vtedy, keď existuje dvojica kolmých vektorov u^α, v^β.

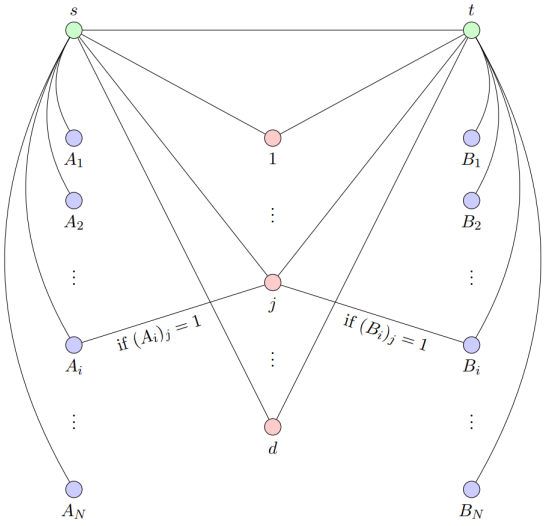
Ak by sme vedeli riešiť Ov v čase $O(N^{2-\varepsilon})$, potom k-Sat by sme vedeli riešiť v čase $O(\left(2^{n/2}\right)^{2-\varepsilon}) = O(2^{(1-\varepsilon/2)n})$ pre ľubovoľné k, čo je v rozpore so SETH. □

Veta 23.3. *Ak platí OVC, potom na výpočet priemeru grafu (maximálnej vzdialenosti medzi dvoma vrcholmi) treba kvadratický čas. Dokonca na 3/2-aproximáciu treba* $\Omega(n^{2-\varepsilon})$.

■ **Dôkaz.** OVC redukujeme na priemer grafu podľa obr. TODO. Vrcholu sú vektory z A a B, indexy $1, \ldots, d$ a dva špeciálne vrcholy. Vrcholy sú pospájané ako na obr. s tým, že hrana medzi vektorom A_i a indexom j vedie vtedy, keď $(A_i)_j = 1$. Platí, že vzdialenosť medzi vektormi A_i a B_j je 3 práve vtedy, keď sú kolmé – v opačnom prípade existuje nejaký index k taký, že $(A_i)_k = (B_j)_k = 1$ a teda cesta A_i—k—B_j má dĺžku 2. Presvedčte sa, že všetky ostatné vzdialenosti sú najviac 2, keďže vrcholy susedia, alebo sú spojené cez s alebo t. Z toho vyplýva, že priemer grafu je 3 práve vtedy, keď existujú dva kolmé vektory A_i, B_j a v opačnom prípade je priemer 2. Ak by existoval $\Omega(n^{2-\varepsilon})$ algoritmus pre 3/2-aproximáciu, tak existuje $\Omega(n^{2-\varepsilon})$ algoritmus pre Ov. □

Veta 23.4. *Ak platí OVC, tak problém* Lcs *sa nedá riešiť v čase* $O(n^{2-\varepsilon})$.

■ **Dôkaz.** (Náznak.) Jednu súradnicu vektoru $a \in A$ nahradíme $0 \mapsto 001$, $1 \mapsto 111$ a pre vektor $b \in B$ nahradíme $0 \mapsto 011$, $1 \mapsto 000$. Všimnite si, že



Obr. 23.1: Redukcia Ov na priemer grafu.

pre ľubovoľnú kombináciu vstupov bude LCS rovné 2, okrem prípadu $(1, 1)$, kde $\mathrm{LCS}(000, 111) = 0$. Takto zakódované súradnice oddeľme dostatočne veľa $(4d)$ znakmi # (tak, aby sa neoplatilo spárovať dve rôzne súradnice). Výsledné stringy budú mať LCS rovné $C = (d-1)4d + 2d$, ak $a \perp b$, v opačnom prípade najviac $C - 2$. Nevýhoda je, že ak a a b nie sú kolmé, LCS môže nadobudnúť veľa rôznych hodnôt medzi $C - 2d$ a $C - 2$. Napravíme to nasledovným trikom: Pridajme každému vektoru $(d+1)$-vú súradnicu rovnú 0, ak je vektor z A, resp. 1, ak je vektor z B (kolmosť medzi vektormi z A a B sa zachová) a definujme špeciálny vektor $s = (0, 0, \ldots, 0, 1)$. Skalárny súčin s a ľubovoľného vektora $b \in B$ je 1 a teda LCS bude presne $C - 2$. Vektory a, b zakódujeme takto:

$$\langle a \rangle * * * \langle s \rangle$$
$$* * * \langle b \rangle * **$$

pričom použijeme dostatočne veľa $(10d^2)$ oddeľovačov $*$ tak, aby sa v LCS vždy oplatilo všetky spárovať. Tzn., v optimálnom riešení budú oddeľovače spárované buď s tými pred b-čkom a b bude spárované s s, a LCS bude $C' = 10d^2 + C - 2$, alebo ak $a \perp b$, oplatí sa oddeľovače spárovať s tými za b-čkom, pričom b spárujeme s a a LCS bude $C' + 2$.

Ako zakódujeme celú množinu vektorov? Vektory z A zakódujeme tak, že zakódujeme dve kópie oddelené dostatočne veľa $(100d^2)$ oddeľovačmi \$:

$$\langle\langle a_1 \rangle\rangle \$\$\$ \langle\langle a_2 \rangle\rangle \$\$\$ \ldots \$\$\$ \langle\langle a_n \rangle\rangle \$\$\$ \langle\langle a_1 \rangle\rangle \$\$\$ \langle\langle a_2 \rangle\rangle \$\$\$ \ldots \$\$\$ \langle\langle a_n \rangle\rangle$$

a vektory z B zakódujeme tak, že podobne zakódujeme jednu kópiu a na začiatok

aj na koniec pridáme $100d^2 \cdot 2n$ oddeľovačov \$:

$$\$\$\$\$\$\langle\langle b_1 \rangle\rangle \$\$\$ \langle\langle b_2 \rangle\rangle \$\$\$ \cdots \$\$\$ \langle\langle b_n \rangle\rangle \$\$\$\$\$\$$$

Ak existuje dvojica $a_i \perp b_j$, LCS bude aspoň $C'' = (2n-1)100d^2 + nC' + 2$ (ak existuje viacero kolmých dvojíc, môže to byť aj viac); naopak, ak taká dvojica neexistuje, LCS bude najviac $C'' - 2$.

Takto sme redukovali Ov na Lcs pre stringy dĺžky $O(nd^2) = O(n \cdot \mathsf{polylog}(n))$ a ak by existoval $O(n^{2-\varepsilon})$ algoritmus pre Lcs, tak existuje aj pre Ov. □

Kapitola 24

Zložitosť počítania

- Koľko sledov dĺžky k vedie medzi s a t?
- Koľko ciest vedie medzi s a t?
- Koľko kostier má daný graf?
- Koľko párovaní má daný graf?
- Koľko 2-farbení má daný graf?
- Koľko riešení má daný 3-SAT?
- Koľko riešení má daný 2-SAT?

Definujme „počítacie“ triedy funkcií $f : \{0,1\}^* \to \mathbb{N}$ analogické P, NP:

Definícia 24.1 (FP, #P). FP *je trieda funkcií* $f : \{0,1\}^* \to \mathbb{N}$ *vypočítateľných v polynomiálnom čase.* $\#\mathsf{P}$ *je trieda funkcií* $f : \{0,1\}^* \to \mathbb{N}$*, pre ktorý existuje polynóm* p *a polynomiálny TS* M *taký, že* $f(x) = |\{y \in \{0,1\}^{p(|x|)} : M(x,y) = 1\}|$*. Inými slovami,* f *je počet akceptačných výpočtov nejakého polynomiálneho NTS.*

Definícia 24.2 (FP). $f \in \mathsf{FP}$*, ak* f *vieme vypočítať v polynomiálnom čase.*

Definícia 24.3 (#P). $f \in \#\mathsf{P}$*, ak* f *je počet akceptačných výpočtov nejakého polynomálneho NTS.*

f je počet „riešení“, pričom riešenia sú poly veľké a vieme ich overiť v poly čase

$\exists$*poly TS* D*, polynóm* p*:* $f(x) = \#\{y \in \{0,1\}^{p(x)} \mid D(x,y) = 1\}$

* $\mathsf{FP} \neq \#\mathsf{P}$?? -* $\mathsf{FP} = \#\mathsf{P}$ ===¿ $\mathsf{P} = \mathsf{NP}$ -* $\mathsf{FP} = \#\mathsf{P}$ ¡=== $\mathsf{P} = \mathsf{NP}$?? (nevie sa)

Definícia 24.4 (#P-úplnosť). *Funkcia* f *je* $\#\mathsf{P}$*-úplna, ak je v* $\#\mathsf{P}$ *a* $\#\mathsf{P} \subseteq \mathsf{FP}^f$.

— * Ak $f \in \mathsf{FP}$, tak $\mathsf{FP}^f = \mathsf{FP}$ * Ak f je #P-úplná a $f \in \mathsf{FP}$, tak $\mathsf{FP} = \#\mathsf{P}$.

Počet sledov dĺžky k vedie medzi s a t? Počet kostier má daný graf? Počet 2-farbení má daný graf? $\in \mathsf{FP}$

Počet ciest vedie medzi s a t? Počet párovaní má daný graf? Počet riešení má daný 2-SAT? sú #P-úplné

#Sat je #P-úplný. väčšina NP-úplných problémov má #P-úplnú počítaciu verziu Funkcia perm na maticiach nad $\mathbb{Z}_2$ je #P-úplná.

Definícia 24.5 (permanent). *Permanent $n \times n$ matice A definujeme ako*

$$\mathrm{perm}(A) = \sum_{\sigma \in S_n} \prod_i A_{i,\sigma(i)}.$$

Veta 24.1 (Valiant). *Funkcia* perm *na maticiach nad $\mathbb{Z}_2$ je #P-úplná.*

Aký je vzťah PH a #P?

Veta 24.2 (Toda). *Všetky problémy v* PH *vieme vyriešiť pomocou jedinej otázky na* #Sat.

$\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}}$.

24.1 Hľadanie jedinečného riešenia

Dôkaz Todovej vety začneme malou odbočkou, a síce otázkou:

Aké ťažké je hľadať jedinečné riešenia?

Vezmime si napr. problém Sat. Ak má formula n premenných, počet riešení (splňujúcich ohodnotení) môže byť ľubovoľné číslo medzi 0 a 2^n. Predstavte si však, že príde niekto a dá nám formulu ϕ, pričom nám sľúbi, že ϕ je buď nesplniteľná, alebo má *práve jedno* jedinečné riešenie. Ak klame, je jedno, ako sa zachová náš algoritmus – či povie áno / nie / zacyklí sa / vybuchne. Tento problém sa volá unique-Sat, USat a otázka znie: pomôže nám táto dodatočná informácia? Je jednoduchšie nájsť riešenie, ak jedinečné? Alebo je naopak USat NP-úplný a teda rovnako ťažký ako všeobecný Sat?

V súčasnosi na túto otázku nepoznáme odpoveď, ale sme veľmi blízko: USat je „skoro“ tak ťažký ako všeobecný Sat. A čo myslíme tým „skoro“? Pri štúdiu NP-úplnosti sme si povedali, že dva problémy budeme považovať za rovnako ťažké, ak vieme redukovať jeden na druhý *deterministickou polynomiálnou many-to-one redukciou*. A v súčasnosi nie je známe, či sa dá Sat redukovať na USat. Čo však vieme, je ako Sat redukovať na USat *pravdepodobnostnou Turingovskou* redukciou, ktorá môže mať malú, jednostrannú chybu:

Veta 24.3 (Valiant, Vazirani). *Existuje pravdepodobnostný polynomiálny algoritmus f taký, že*

- *ak $\phi \in$ Sat, tak $\Pr[f(\phi) \in \mathrm{USat}] \geq 1/8n$, ale*

- *ak* $\phi \notin$ SAT, *tak ani* $f(\phi) \notin$ SAT.

Inak povedané: Platí, že $\mathsf{NP} \subseteq \mathsf{P}^{\mathrm{SAT}}$ a nevieme, či $\mathsf{NP} \subseteq \mathsf{P}^{\mathrm{USAT}}$, ale dokážeme si, že

$$\mathsf{NP} \subseteq \mathsf{RP}^{\mathrm{USAT}}.$$

Alebo ešte inak: Neveríme, že USAT sa dá riešiť (deterministicky) v polynomiálnom čase, pretože potom by sa SAT dal riešiť pravdepodobnostne v polynomiálnom čase a potom by $\mathsf{NP} = \mathsf{RP}$.

Lema 24.1 (Izolačná lema, Valiant, Vazirani). *Nech* $\mathcal{H}_{n,k}$ *je trieda po dvoch nezávislých hašovacích funkcií* $h : \{0,1\}^n \to \{0,1\}^k$ *a* $S \subseteq \{0,1\}^n$, $\frac{1}{4} \cdot 2^k \leq |S| \leq \frac{1}{2} \cdot 2^k$. *Potom* $\Pr_{h \in_R \mathcal{H}_{n,k}}[\exists!\, x \in S : h(x) = 0^k] \geq 1/8$.

■ **Dôkaz.** Označme p pravdepodobnosť, že ak vyberiem náhodnú hešovaciu funkciu z $\mathcal{H}$, tak sa nejaký (ľubovoľný) fixný prvok x zahešuje na nulu: $p = \Pr_{h \in_R \mathcal{H}_{n,k}}[h(x) = 0^k] = 1/2^k$. Nech N je náhodná premenná označujúca počet prvkov $x \in S$, ktoré sa zahešujú na nulu. Zjavne $E[N] = |S|p \in [\frac{1}{4}, \frac{1}{2}]$.

Platí: $\Pr[N = 0] \leq 1 - |S|p + \binom{|S|}{2}p^2$ (princíp inklúzie, exklúzie) a $\Pr[N \geq 2] \leq \binom{|S|}{2}p^2$ (union bound), odkiaľ $\Pr[N = 1] \geq |S|p(1 - |S|p) \geq 1/8$. □

■ **Dôkaz vety.** Zvolíme náhodné $k \in_R \{2, \ldots, n+1\}$ a $h \in_R \mathcal{H}_{n,k}$ (s pravdepodobnosťou $1/n$ bude $2^{k-2} \leq |S| \leq 2^{k-1}$ a vtedy s pravdepodobnosťou $1/8$ existuje jediné x: $h(x) = 0^k$). Nech teda $\tau(x, y)$ hovorí „$h(x) = 0^k$“ (túto formulu zostrojíme buď priamo podľa $\mathcal{H}$, alebo v najhoršom prípade Cook-Levinovou konštrukciou). Výsledkom je formula $\psi \equiv \phi(x) \wedge \tau(x, y)$. □

24.2 Parita počtu riešení

Definícia 24.6 ($\oplus$P). $L \in \oplus\mathsf{P}$, *ak existuje poly-time NTS* M *taký, že* $x \in L$ *práve vtedy, ak* $M(x)$ *má nepárny počet akceptačných výpočtov. Definujeme* $\oplus\phi(x)$ *ako pravdivú, ak* $\phi(x)$ *má nepárny počet splňujúcich ohodnotení a* $\oplus$SAT *ako jazyk pravdivých* $\oplus\phi$ *formúl (*ϕ *nemusí byť v CNF).*

Zatiaľčo pri USAT je otvorený problém zlepšiť pravdepodobnosť úspechu čo i len na konštantu, pri $\oplus$SAT to pôjde.

Označme $\#\phi$ počet splňujúcich ohodnotení ϕ. Pre ϕ, ψ vieme vyrobiť formuly $[\phi \cdot \psi](x, y) \equiv \phi(x) \wedge \psi(y)$ a (predpokladajme, že ϕ má n premenných a ψ $m \leq n$ premenných) $[\phi + \psi](z) \equiv [z_0 \wedge \phi(z) \wedge \bigwedge_{i=m+1}^{n} \neg z_i)] \vee [\neg z_0 \wedge \psi(z) \wedge \bigwedge_{i=1}^{m} \neg z_i)]$ také, že $\#[\phi \cdot \psi] = \#\phi \cdot \#\psi$ a $\#[\phi + \psi] = \#\phi + \#\psi$.

Nech 1 je formula s jediným splňujúcim ohodnotením. Všimnime si, že $[\bigoplus_x \phi(x)] \wedge [\bigoplus_y \psi(y)] \iff \bigoplus_{x,y} [\phi \cdot \psi](x, y)$ a $\neg \bigoplus_x \phi(x) \iff \bigoplus_{x,z} [\phi + 1](x, z)$.

Lema 24.2. *Pre ľubovoľné* k, m *existuje pravdepodobnostný polynomiálny algoritmus* f *taký, že*

- *ak* $\phi \in \Sigma_k$SAT, *tak* $\Pr[f(\phi) \in \oplus\mathrm{SAT}] \geq 1 - 1/2^m$, *ale*

- *ak* $\phi \notin \Sigma_k$SAT*, tak ani* $Pr[f(\phi) \notin \text{SAT}] \leq 1/2^m$.

Inými slovami $\mathsf{PH} \subseteq \mathsf{BPP}^{\oplus\text{SAT}}$.

■ **Dôkaz.** Pre NP vezmeme formulu ϕ a použijeme redukciu z vety 24.1 $R = O(nm)$-krát; výsledná formula bude zakódovanie ich OR-u (popísané vyššie). Ak redukcia nefungovala s pp. $1 - 1/8n$, tak R-krát zopakovaná nefunguje s pp. $\leq 2^{-m}$. Vďaka uzavretosti $\oplus$P na NOT nám prejde aj coNP.

Pre $c > 1$ bude dôkaz prebiehať indukciou. Nech $\phi = \exists x : \psi(x)$, x pozostáva z n premenných. Z indukčného predpokladu existuje pravdepodobnostná redukcia, ktorá pre každé x zostrojí $\oplus$SAT formulu $\beta(x) \equiv \bigoplus_z \rho(z, x)$ ekvivalentnú $\psi(x)$ s pravdepodobnosťou aspoň $1 - 2^{-(m+1)}$. Ak spustíme Valiantovu-Vaziraniho redukciu $R = O(nm)$-krát, dostaneme $\alpha \equiv \bigvee_{j=1}^{R}(\bigoplus_{x,y} \tau_j(x, y) \wedge \beta(x))$. Ak je $\exists x : \beta(x)$ pravdivá, Pr[αje pravdivá] $\geq (1 - 1/8n)^R \geq 1 - 2^{-(m+1)}$. Inak je α nepravdivá.

Z IP β je $\oplus$SAT inštancia a α vieme do tohto tvaru upraviť. Chyba môže nastať pri prevode $\psi(x)$ na β a pri Valiantovej-Vaziraniho redukcií; pravdepodobnosť je však najviac $2 \times 2^{-(m+1)}$. □

Náznak iného dôkazu: Začnime s tým, že $\mathsf{NP} \subseteq \mathsf{BPP}^{\oplus\mathsf{P}}$. Dá sa ukázať, že $\oplus\mathsf{P}^{\oplus\mathsf{P}} = \oplus\mathsf{P}$ a že keby $\mathsf{NP} \subseteq \mathsf{BPP}$ (čo nepredpokladáme), tak celá $\mathsf{PH} \subseteq \mathsf{BPP}$. Všetky tieto výsledky relativizujú, tzn. platia, aj keby sme všetkým strojom pridali (to isté) orákulum:

pôvodná veta	relativizovaná
$\mathsf{NP} \subseteq \mathsf{BPP}^{\oplus\mathsf{P}}$	$\mathsf{NP}^{\oplus\mathsf{P}} \subseteq \mathsf{BPP}^{\oplus\mathsf{P}^{\oplus\mathsf{P}}}$
$\mathsf{NP} \subseteq \mathsf{BPP} \Rightarrow \mathsf{PH} \subseteq \mathsf{BPP}$	$\mathsf{NP}^{\oplus\mathsf{P}} \subseteq \mathsf{BPP}^{\oplus\mathsf{P}} \Rightarrow \mathsf{PH}^{\oplus\mathsf{P}} \subseteq \mathsf{BPP}^{\oplus\mathsf{P}}$

Takže $\mathsf{NP}^{\oplus\mathsf{P}} \subseteq \mathsf{BPP}^{\oplus\mathsf{P}^{\oplus\mathsf{P}}} = \mathsf{BPP}^{\oplus\mathsf{P}}$ a z $\mathsf{NP}^{\oplus\mathsf{P}} \subseteq \mathsf{BPP}^{\oplus\mathsf{P}}$ vyplýva $\mathsf{PH} \subseteq \mathsf{PH}^{\oplus\mathsf{P}} \subseteq \mathsf{BPP}^{\oplus\mathsf{P}}$.

24.3 Todova veta

Veta 24.4 (Toda). $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}}$.

Lema 24.3. *Existuje (deterministická) poly-time transformácia T, ktorá ku každej formule α vyrobí $\beta = T(\alpha, 1^\ell)$, pričom*

- *ak* $\alpha \in \oplus$SAT*, tzn.* $\#\alpha \equiv -1 \pmod 2$*, tak* $\#\beta \equiv -1 \pmod{2^{\ell+1}}$*, ale*
- *ak* $\alpha \notin \oplus$SAT*, tzn.* $\#\alpha \equiv 0 \pmod 2$*, tak* $\#\beta \equiv 0 \pmod{2^{\ell+1}}$.

■ **Dôkaz Todovej vety.** Nech f je redukcia z Lemy 24.2 s $m = 2$, ktorá na vstupe dostane kvantifikovanú formulu ψ a náhodný reťazec r dĺžky R. Nech T je redukcia z Lemy 24.3 s $\ell = R + 2$ bežiaca v čase $\mathsf{poly}(R, |f(\psi)|)$. Všimnime si teraz hodnotu $\sum_{r \in \{0,1\}^R} \#T(f(\psi, r)) \pmod{2^{\ell+1}}$.

Ak je ψ pravdivá, aspoň 3/4 členov je -1, takže súčet leží medzi -2^R a $-\lceil \frac{3}{4} \times 2^R \rceil$. Na druhej strane, ak je ψ nepravdivá, súčet je medzi $-\lceil \frac{1}{4} \times 2^R \rceil$ a 0. Stačí teda pomocou Cookovej-Levinovej konštrukcie zostrojiť formulu $\Gamma(r, y, z)$ takú, že

$$\#\Gamma = \sum_r \#T(f(\psi, r)) \pmod{2^{\ell+1}}.$$

$\Gamma(r, y, z)$ zostrojíme tak, aby bola pravdivá práve vtedy, keď y splňuje $T(f(\psi, r), 1^\ell)$, pričom z sú pomocné premenné kódujúce výpočet. □

■ **Dôkaz lemy.** Nech $g(x) = 4x^3 + 3x^4$, $g^0(x) = x$, $g^{i+1}(x) = g(g^i(x))$. Všimnime si, že ak $x \equiv -1/0 \pmod{2^{2^i}}$, tak $g(x) \equiv -1/0 \pmod{2^{2^{i+1}}}$.

To znamená, že ak $\#\tau \equiv -1/0 \pmod{2^{2^i}}$, tak aj $\#[4\tau^3 + 3\tau^4] \equiv -1/0 \pmod{2^{2^{i+1}}}$, kde τ^3 je skratka pre formulu $[\tau \cdot [\tau \cdot \tau]]$.

Nech $\psi_0 = \alpha$, $\psi_{i+1} = 4\psi_i^3 + 3\psi_i^4$ a $\beta = \psi_{\lceil \log(\ell+1) \rceil}$. Formula β bude iba $\exp(O(\log \ell)) = \mathsf{poly}(\ell)$-krát dlhšia. □

Kapitola 25

Derandomizácia a dolné odhady

Veta 25.1 (Karp-Lipton). *Ak* $\mathsf{NP} \subseteq \mathsf{P/poly}$*, tak* $\mathsf{PH} = \Sigma_2^{\mathsf{P}}$.

Veta 25.2 (Meyer). *Ak* $\mathsf{EXP} \subseteq \mathsf{P/poly}$*, tak* $\mathsf{EXP} = \Sigma_2^{\mathsf{P}}$.

Veta 25.3. *Ak* $\mathsf{NEXP} \subseteq \mathsf{P/poly}$*, tak* $\mathsf{NEXP} = \mathsf{EXP} = \Sigma_2^{\mathsf{P}}$.

Veta 25.4. *Ak* $\text{PIT} \in \mathsf{P}$*, a* $\textsc{Perm} \in \mathsf{AlgP/poly}$*, tak* $\mathsf{P}^{\textsc{Perm}} \subseteq \mathsf{NP}$.

■ **Dôkaz.** * nech $L \in \mathsf{P}^{\textsc{Perm}}$, M akc. L v čase $m = n^d$ * zostrojíme N: * N si tipne algebraické obvody $C_1, \ldots, C_m$ * overí, že C_i rieši Perm na maticiach $i \times i$ * simuluje M s pomocou $\{C_i\}$ * overovanie je induktívne: pre $t \times t$ maticu je

$$\operatorname{perm} A = \sum_{i=1}^{t} a_{1i} \operatorname{perm} A_{1,i}$$

* $A_{i,j}$ je A s vynechaným i-tym riadkom a j-tym stĺpcom * ak sme už overili C_{t-1}, nahradením $C_t(X)$ za $\operatorname{perm} X$ dostávame identitu s t^2 vstupmi * tú vieme overiť vďaka $\text{PIT} \in \mathsf{P}$. □

===

Veta 25.5. *Ak* $\text{PIT} \in \mathsf{P}$*, tak* $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$ *alebo* $\textsc{Perm} \notin \mathsf{AlgP/poly}$.

■ **Dôkaz.** * nech $\text{PIT} \in \mathsf{P}$, $\mathsf{NEXP} \subseteq \mathsf{P/poly}$ a $\textsc{Perm} \in \mathsf{AlgP/poly}$ * $\mathsf{NEXP} = \mathsf{EXP} = \Sigma_2^{\mathsf{P}} \subseteq \mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}} \subseteq \mathsf{P}^{\textsc{Perm}} \subseteq \mathsf{NP}$ * spor s vetou o nedeterministickej časovej hierarchii. □

===

výsledok „3. generácie“ * časová hierarchia, Hartmanis, Stearns – 1965 * nedeterministická časová hierarchia, Cook – 1972 * permanent a #P-úplnosť, Valiant – 1979 * Karp-Lipton-Meyer – 1980

* Todova veta – 1991 * Impagliazzo, Kabanetz, Wigderson – 2001

Časť IX

Apendix

Dodatok A

Užitočné znalosti z matematiky

A.1 Aproximácie, odhady, nerovnosti

$$\left(1+\frac{1}{n}\right)^n < e < \left(1+\frac{1}{n}\right)^{n+1}$$

$$\left(1-\frac{1}{n}\right)^n < 1/e < \left(1+\frac{1}{n}\right)^{n-1}$$

Zjavne $(n/2)^{n/2} \leq n! \leq n^n$ (polovica činiteľov je aspoň polovica), takže $\ln n! = \Theta(n \log n)$. Presnejší odhad dáva Stirlingova aproximácia:

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

$$\ln n! = n \ln n - n + O(\ln n)$$

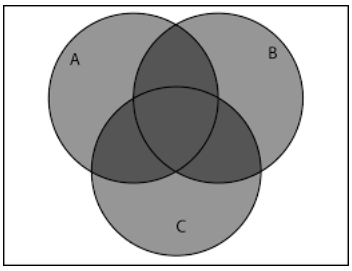
A.2 Základy pravdepodobnosti

Veta A.1 (Booleova nerovnosť (Union bound)). *Pravdepodobnosť zjednotenia je najviac súčet pravdepodobností:*

$$\Pr[\bigcup A_i] \leq \sum_i \Pr[A_i].$$

A.3 Pravdepodobnostná metóda

Ak chceme dokázať existenciu objektu s nejakými vlastnosťami, môžeme namiesto toho skúsiť (zdola) odhadnúť pravdepodobnosť, že náhodný objekt má dané vlastnosti. Ak je pravdepodobnosť nenulová, taký objekt musí existovať.



Obr. A.1: Pravdepodobnosť $A \cup B \cup C$ sa dá vyjadriť presne podľa princípu inklúzie a exklúzie. Každopádne sa však dá zhora ohraničiť súčtom jednotlivých pravdepodobností A, B a C.

Veta A.2 (Priemerovací princíp). *Zjavne pre ľubovoľnú postupnosť je minimum ≤ priemer ≤ maximum. To ale znamená, že ak priemer je p, potom existuje člen s hodnotou ≥ p. Ak X je náhodná premenná nad konečnou množinou a $E[X] \geq \mu$, potom $X \geq \mu$ s nenulovou pravdepodobnosťou. Ak jav A závisí od náhodných premenných X, Y, tak ak $\Pr_{X,Y}[A(X,Y)] \geq \mu$, tak existuje konkrétna hodnota x, pre ktorú $\Pr_Y[A(x,Y)] \geq \mu$.*

A.4 Odhady chvostov distribúcií

Veta A.3 (Markovova nerovnosť). *Nech X je náhodná premenná, ktorá nadobúda iba nezáporné hodnoty. Potom pre každé $k > 0$ platí:*

$$\Pr[X \geq k] \leq E[X]/k \qquad \Pr[X \geq k\mu] \leq 1/k$$

■ **Dôkaz.**

$$\mathrm{E}[X] = \sum_t t \cdot \Pr[X = t] \geq \sum_{t \geq k} t \cdot \Pr[X = t] \geq \sum_{t \geq k} k \cdot \Pr[X = t] = k \Pr[X \geq k]$$

□

Veta A.4 (Čebyševova nerovnosť). *Pre ľubovoľné $k > 0$ platí*

$$\Pr[|X - E[X]| \geq k] \leq Var[X]/k^2 \qquad \Pr[|X - E[X]| \geq k\sigma] \leq 1/k^2$$

■ **Dôkaz.** Stačí si uvedomiť, že

$$\Pr[|X - \mathrm{E}[X]| \geq k] = \Pr[(X - \mathrm{E}[X])^2 \geq k^2]$$

a použiť Markovovu nerovnosť na premennú $Y = (X - \mathrm{E}[X])^2$ ($\mathrm{E}[Y] = \mathrm{V}ar[X]$).
□

Veta A.5 (Černofova nerovnosť). *Nech $X_1, \ldots, X_n$ sú nezávislé 0–1 náhodné premenné (tzv. Poissonove pokusy), pričom $p_i = \Pr(X_i = 1)$, $X = \sum_i X_i$ a $\mu = E[X] = \sum p_i$. Potom*

- $\forall \delta > 0 : \Pr(X \geq (1+\delta)\mu) < (e^\delta/(1+\delta)^{(1+\delta)})^\mu$
- $\forall 0 < \delta \leq 1 : \Pr(X \geq (1+\delta)\mu) \leq e^{-\mu\delta^2/3}$
- $\forall R \geq 6\mu : \Pr(X \geq R) \leq 2^{-R}$

■ **Dôkaz.** Opäť použijeme Markovovu nerovnosť, avšak tentokrát na výraz $\exp(tX)$, kde t je vhodná konštanta, ktorú určíme až na konci. Vďaka monotónnosti exp platí

$$\Pr(X \geq (1+\delta)\mu) = \Pr(e^{tX} \geq e^{t(1+\delta)\mu}) \leq \frac{\mathrm{E}[e^{tX}]}{e^{t(1+\delta)\mu}}.$$

Keďže jednotlivé X_i a teda aj e^{tX_i} sú nezávislé, môžeme písať

$$\mathrm{E}[e^{tX}] = \mathrm{E}[e^{\sum_i tX_i}] = \mathrm{E}[\prod_i e^{tX_i}] = \prod_i \mathrm{E}[e^{tX_i}] = \prod_i (p_i e^t + (1-p_i)) = \prod_i (1 + p_i(e^t - 1))$$

Keďže $1 + y < e^y$, $1 + p_i(e^t - 1) < \exp(p_i(e^t - 1))$, tzn.

$$\mathrm{E}[e^{tX}] < \prod_i e^{p_i(e^t-1)} = e^{\sum_i p_i(e^t-1)} = e^{(e^t-1)\mu}$$

a teda

$$\Pr(X \geq (1+\delta)\mu) < \frac{e^{(e^t-1)\mu}}{e^{t(1+\delta)\mu}} = e^{(e^t-1-t-t\delta)\mu}$$

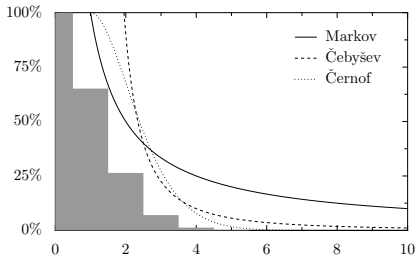
Táto nerovnosť platí pre ľubovoľné t; na záver už len zvolíme to najlepšie. Zderivovaním zistíme, že $e^t - 1 - t(1+\delta)$ nadobúda minimum pre $t = \ln(1+\delta)$. Dosadením a zjednodušením dostaneme prvú nerovnosť. □

A.5 Polynómy

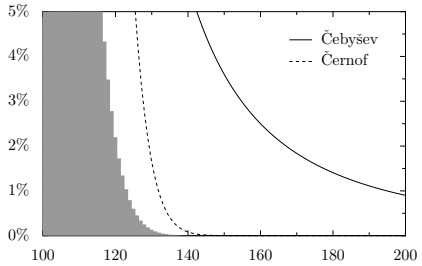
Veta A.6. *Nenulový polynóm stupňa n (nad ľubovoľným poľom) má najviac n koreňov (vrátane násobností). Ak sa teda dva polynómy stupňa $\leq$ rovnajú v $n+1$ bodoch, musia byť rovnaké.*

■ **Dôkaz.** Ak α je koreň p, potom $(x - \alpha)$ delí p (vo všeobecnosti $p \bmod (x - \alpha) = p(\alpha)$) a $p/(x - \alpha)$ má stupeň o jedna menší. Veta je teda dokázaná indukciou.

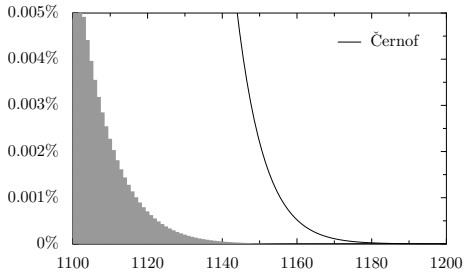
Ak sa p, q stupňa $\leq n$ rovnajú v $n+1$ bodoch, potom $p - q$ je stupňa $\leq n$ a má $n+1$ koreňov, takže to musí byť nulový polynóm. □



(a) $Bin(10, 0.2)$, $E[X] = 2$, z Markovovej nerovnosti $\Pr[X \geq 4] \leq 1/2$



(b) $Bin(2000, 0.1)$, $E[X] = 200$, $\Pr[X \geq 250] < 0.5\%$



(c) $Bin(10000, 0.1)$, $E[X] = 100$, $\Pr[X \geq 140] < 0.1\%$

Obr. A.2: Odhady chvostov distribúcií

Lema A.1 (Schwartz-Zippel). *Označme $\mathbb{F}_{d,k}$ množinu polynómov o k premenných celkového stupňa d nad konečným poľom $\mathbb{F}_q$. Nenulový polynóm z $\mathbb{F}_{d,k}$ má najviac dq^{k-1} koreňov, to znamená d/q-tina všetkých bodov z $\mathbb{F}_q^k$. Z toho vyplýva, že dva rôzne polynómy z $\mathbb{F}_{d,k}$ v zhodujú na najviac d/q-tine vstupov. Z pohľadu pravdepodobnosti lema hovorí, že pre dva rôzne polynómy $p, q \in \mathbb{F}_{d,k}$ je*

$$\Pr_{\vec{x}}[p(\vec{x}) = q(\vec{x}) \mid p \neq q] \leq d/q,$$

resp. pre nenulový polynóm $p \in \mathbb{F}_{d,k}$ máme

$$\Pr_{\vec{x}}[p(\vec{x}) = 0 \mid p \neq 0] \leq d/q.$$

■ **Dôkaz.** Lemu dokážeme indukciou vzhľadom na k: pre $k = 1$ máme polynómy jednej premennej a tie majú najviac d koreňov (ak d je ich stupeň), pretože $f(\alpha) = 0$ práve vtedy, keď $(X - \alpha) \setminus f(X)$. Teda pre $k = 1$ naozaj $\Pr_x[f(x) = 0 \mid f \not\equiv 0] \leq d/q$.

Majme teraz polynóm $f(X_1, \ldots, X_k) \in \mathbb{F}_{d,k}$. Nech m je najvyššia mocnina X_1 v f; ak vyjmeme z každého členu mocninu X_1, dostaneme

$$f(X_1, \ldots, X_k) = \sum_{i=0}^{m} X_1^i \cdot g_i(X_2, \ldots, X_k).$$

Všimnite si, že ak za $X_2, \ldots, X_k$ dosadíme nejaké konkrétne hodnoty $\alpha_2, \ldots, \alpha_k$, dostaneme polynóm $h(X_1) = \sum_{i=0}^{m} \beta_i X_1^i$ jednej premennej stupňa najviac m (kde $\beta_i = g_i(\alpha_2, \ldots, \alpha_k)$).

Nech $(\alpha_1, \ldots, \alpha_k)$ je koreň f; toto môže nastať iba dvoma spôsobmi:

- buď je $(\alpha_2, \ldots, \alpha_k)$ je koreňom g_m, alebo
- dosadením $\alpha_2, \ldots, \alpha_k$ dostaneme polynóm $h(X_1) = f(X_1, \alpha_2, \ldots, \alpha_k)$ a teda α_1 je koreňom $h(X_1)$.

Všimnite si, že g_m je stupňa najviac $d-m$ a teda z indukčného predpokladu nastane prvá možnosť s pravdepodobnosťou najviac $(d-m)/q$; v druhom prípade máme polynóm jednej premennej stupňa m a α_1 je jeho koreňom s pravdepodobnosťou najviac m/q. Odtiaľ

$$\Pr_{\vec{x}}[f(\vec{x}) = 0 \mid f \not\equiv 0] \leq (d-m)/q + m/q = d/q. \qquad \square$$

A.6 Lineárne programovanie

Dodatok B

Zhrnutie

P

Problémy riešiteľné deterministicky v polynomiálnom čase, napr. na Turingovom stroji, alebo na RAM. Tiež problémy riešiteľné (logspace) uniformnými obvodmi polynomiálnej veľkosti. $\mathsf{P} = \mathsf{AL}$, teda sú to tiež problémy riešiteľné v logaritmickom priestore alternujúcim TS.

Problémy CVP (circuit value problem – vyhodnotiť daný boolovský obvod) či LP (lineárne programovanie) sú P-úplné (pri logspace redukcii).

NP a coNP

NP sú problémy riešiteľné nedeterministicky v polynomiálnom čase. Ekvivalentne, sú to také problémy, že ak nám niekto prezradí riešenie, vieme ho efektívne (v P) overiť. Problémy v NP majú pravdepodobnostne overiteľné dôkazy, $\mathsf{NP} = \mathsf{PCP}[O(\log n), O(1)]$. coNP tvoria komplementy jazykov v NP.

NP-úplné sú napr. Sat, ILP (celočíselné lineárne programovanie), Clique, Vertex-Cover, Hamilton-Circuit, 3-Coloring. Zistiť, či je daná formula tautológia, je coNP-úplný problém.

PSPACE

Problémy riešiteľné v polynomiálnom priestore, napr. na TS alebo RAM. Podľa Savitchovej vety je PSPACE = NPSPACE (nedeterministický polynomiálny priestor). Podľa Immerman-Szelepcsényiho vety je priestor uzavretý na komplement, teda PSPACE = coPSPACE. Ďalšia charakterizácia je PSPACE = AP – sú to práve problémy riešiteľné v polynomiálnom čase alternujúcim TS; a PSPACE = IP (interaktívne dokazovacie systémy).

PSPACE-úplné sú QBF (quantified boolean formula), Geography, Sokoban, Reversi, ekvivalencia regulárnych výrazov, atď.

EXP a NEXP

EXP sú problémy riešiteľné v exponenciálnom čase (t.j. $O(2^{\mathsf{poly}(n)})$). EXP = APSPACE – polynomiálny priestor + alternácie. NEXP to isté nedeterministicky. NEXP sú problémy, ktoré majú interaktívne dokazovacie systémy s aspoň dvoma provermi, NEXP = MIP, resp. pravdepodobnostne overiteľné dôkazy, NEXP = $\mathsf{PCP}[\mathsf{poly}(n), \mathsf{poly}(n)] = \mathsf{PCP}[\mathsf{poly}(n), O(1)]$.

Hry ako dáma, šach (zovšeobecnený), či Go (s japonským pravidlom ko) sa ukázali ako EXP-úplné. Rôzne varianty NP-úplných problémov, kde vstup zapíšeme úsporne sú NEXP-úplné, napr. úsporný 3SAT: na vstupe je obvod, ktorý kóduje exponenciálne dlhú formulu – je splniteľná?

PH

Polynomiálna hierarchia je hierarchia tried medzi P a PSPACE. $\mathsf{P} = \Sigma_0^\mathsf{P} \subseteq \mathsf{NP} = \Sigma_1^\mathsf{P} \subseteq \Sigma_2^\mathsf{P} \subseteq \cdots \subseteq \mathsf{PH} \subseteq \mathsf{PSPACE}$, resp. $\mathsf{P} = \Pi_0^\mathsf{P} \subseteq \mathsf{coNP} = \Pi_1^\mathsf{P} \subseteq \Pi_2^\mathsf{P} \subseteq \cdots \subseteq \mathsf{PH} \subseteq \mathsf{PSPACE}$. Definovať k-ty stupeň hierarchie môžeme cez alternujúce TS, ktoré iba $(k-1)$-krát alternujú alebo cez TS s orákulom: $\Sigma_{k+1}^\mathsf{P} = \mathsf{NP}^{\Sigma_k^\mathsf{P}}$ alebo cez polynomiálne relácie: $L \in \Sigma_k^\mathsf{P}$, ak $y \in L$ práve vtedy, keď $\exists x_1 \forall x_2 \cdots (\exists/\forall) x_k R(y, x1, ..., xk)$. $\Pi_{k+1}^\mathsf{P} = \mathsf{coNP}^{\Sigma_k^\mathsf{P}} = \mathsf{co}\Sigma_{k+1}^\mathsf{P}$.

Hoci PH samotná asi nemá úplné problémy (inak skolabuje), na každej úrovni je úplný problém zistiť, či je kvantifikovaná formula s k kvantifikátormi pravdivá. Vieme, že BPP patrí do druhej úrovne polynomiálnej hierarchie a ak by sa SAT dal riešiť neuniformnými polynomiálnymi obvodmi (ak $\mathsf{NP} \subseteq \mathsf{P/poly}$), hierarchia skolabuje.

P/poly

Neuniformné triedy obvodov polynomiálnej veľkosti. Pre každú veľkosť vstupu máme iný obvod. Ekvivalentná definícia je cez TS, ktorým pre každú veľkosť vstupu môžeme dať „radu“ polynomálnej veľkosti. Rada závisí iba od dĺžky vstupu, nie od vstupu samotného.

P/poly je trochu čudná trieda v tom, že obsahuje aj nerozhodnuteľné jazyky (odpovede môžeme zadrátovať do obvodu vďaka neuniformnosti). Avšak veríme, že ani neuniformné polynomiálne obvody nedokážu riešiť SAT (inak skolabuje PH). Neuniformnosť je viac ako náhodnosť: $\mathsf{BPP} \subseteq \mathsf{P/poly}$.

AC a NC

NC zachytáva pojem „efektívne paralelné algoritmy“. NC je trieda problémov, ktoré sa dajú na PRAM riešiť paralelne v polylogaritmickom čase s polynomiálnym počtom procesorov. Ekvivalentná definícia cez obvody: NC = AC sú problémy riešiteľné (uniformnými) obvodmi (s AND/OR/NOT) polynomiálnej

veľkosti a polylogaritmickej hĺbky. Obvody v NC majú AND/OR hradlá s dvoma vstupmi, v AC môžu mať neobmedzene veľa vstupov. Pre AC a NC môžeme zadefinovať hierarchie, pričom na k-tej úrovni povolíme obvody hĺbky $O(\log^k n)$. Zrejme $\mathsf{AC}^0 \subseteq \mathsf{NC}^1 \subseteq \mathsf{AC}^1 \subseteq \mathsf{NC}^2 \subseteq \mathsf{AC}^2 \subseteq \cdots$. Podľa Barringtonovej vety NC^1 sú práve problémy riešiteľné vetviacimi programmi šírky 5. Vieme, že $\mathsf{AC}^0 \neq \mathsf{NC}^1$. Predpokladá, sa že hierarchia je striktná celá, ale je to otvorený problém.

AC^0 obsahuje napr. neregulárny jazyk „$a^n b^n$“, sčítanie celých čísel, násobenie boolovských matíc, ale nedokáže spočítať paritu (regulárny jazyk!), či majoritu (je na vstupe viac 0 alebo 1?). NC^1 obsahuje všetky regulárne jazyky, súčin celých čísel, AC^1 vie počítať tranzitívny uzáver matice a teda napr. či v danom grafe existuje cesta medzi dvoma vrcholmi. AC^1 vie rozoznávať všetky bezkontextové jazyky.

L a NL

Deterministický, resp. nedeterministický logaritmický priestor. $\mathsf{NL} = \mathsf{coNL}$. $\mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{NL} = \mathsf{coNL} \subseteq \mathsf{AC}^1$.

Hľadanie cesty v orientovanom grafe je NL-úplný problém. Úloha sa pre neorientované grafy dá riešiť deterministicky v L (ťažký výsledok založený na derandomizovaní náhodnej prechádzky po grafe).

BPP

Problémy riešiteľné efektívnymi pravdepodobnostnými algoritmami, ktoré sa môžu pomýliť (jedným aj druhým smerom) s pravdepodobnosťou 1/3 (dá sa zlepšiť na exponenciálne malú pravdepodobnosť). $\mathsf{BPP} \subseteq \mathsf{P/poly}$ a zároveň $\mathsf{BPP} \subseteq \Sigma_2^{\mathsf{P}}$. Predpokladá sa, že $\mathsf{P} = \mathsf{BPP}$, teda každý BPP problém sa dá efektívne derandomizovať (otvorený problém).

O známom probléme testovania prvočíselnosti sa nakoniec ukázalo, že sa dá riešiť deterministicky v polynomiálnom čase (hoci v praxi sa dodnes používajú rýchlejšie pravdepodobnostné algoritmy). Problém testovania, či sa dva polynómy viacerých premenných (nad nejakým konečným poľom) rovnajú, je v BPP (stačí polynómy vyhodnocovať na náhodných vstupoch). Zatiaľ sa nevie, či sa dá efektívne derandomizovať.

#P

Trieda funkcií (nie rozhodovacích problémov) f, ktoré predstavujú počet riešení problémov v NP (počet akceptačných výpočtov NTS). Prekvapujúco, $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}} \subseteq \mathsf{PSPACE}$.

Problém #Sat (počet splňujúcich ohodnotení), či #Perfect-Matching (počítanie perfektných párovaní) je #P-úplné.