Týden 13

Přednáška - první část

Nejprve jsme si připomněli strukturu písemky u závěrečné zkoušky. (Hlavně jsem zdůraznil druhou část.)
Pak jsme se vrátili k některým věcem z dřívějška a přidali nové věci, jak je zachyceno níže. (Text neodpovídá přesnému průběhu přednášky, něco je ze cvičení, apod.)

Třída PTIME

Připomněli jsme si definici třídy PTIME. Přitom jsem zdůraznil, že všechny problémy ve studijním textu (nejen ty, které jsou v PTIME) je třeba důkladně promyslet – pak nemůže být pro nikoho problémem u zkoušky nějaký požadovaný problém přesně definovat a uvést příklady instancí s pozitivní odpovědí a instancí s negativní odpovědí.

Nedeterministické algoritmy a jejich složitost; třída NPTIME

Ujasnili jsme si pojem nedeterministického algoritmu (Turingova stroje) a definici toho, co to znamená, že nějaký nedeterministický Turingův stroj M rozhoduje daný problém P.
Také jsme přímočaře rozšířili pojem (časové) složitosti na nedeterministické stroje a definovali jsme třídu NPTIME.
Ukázali jsme si (nedeterministické) algoritmy prokazující, že problém SAT (splnitelnost booleovských formulí) a IS (Independent Set, rozhodovací verze problému nezávislé množiny v grafu) jsou v NPTIME.

Polynomiální převeditelnost; NP-úplné problémy

Již známý pojem převeditelnosti mezi problémy jsme využili v definici polynomiální převeditelnosti mezi problémy. (Příslušný převádějící algoritmus musí mít polynomiální časovou složitost, tedy časovou složitost omezenou polynomem.)
Všimli jsme si, jak může prokázaná polynomiální převeditelnost mezi problémy pomoci v určování (ne)příslušnosti k PTIME či NPTIME.
Definovali jsme pojem NP-úplného problému; problémy SAT, 3-SAT, HC, HK, 3-CG a IS jsou příklady NP-úplných problémů.

PSPACE, NPSPACE, PSPACE-úplnost

Uvědomili jsme si nejprve, že např. pro zjištění toho, zda Bílý má nějakou strategii ve hře ŠACHY, která mu zaručuje vítězství v 200 tazích (rozumí se, že Bílý táhne maximálně 200-krát), bychom uměli celkem přímočaře sestavit algoritmus; např. zavoláme

MaBilyVS(VychoziPozice, Bílý, 200), kde

> MaBilyVS(Pozice, NaTahu, Limit):
>
> if ((Pozice, NaTahu) představuje mat Černému) return ANO;
> if ((Pozice, NaTahu) představuje pat nebo mat Bílému nebo Limit=0) return NE;
> if (NaTahu=Bílý) {Postupně pro každý tah Bílého v Pozice zavolej MaBilyVS(Pozice', Černý, Limit), kde Pozice' vznikne z Pozice provedením příslušného tahu; když je v nějakém případě vráceno ANO, tak return ANO, jinak return NE};
> if (NaTahu=Černý) {Postupně pro každý tah Černého zavolej MaBilyVS(Pozice', Bílý, Limit-1); když je ve všech případech vráceno ANO, tak return ANO, jinak return NE}.

Snadno si ovšem spočteme, že odpovědi bychom se od tohoto algoritmu nedočkali, ale není to tím, že by přetekla paměť. Je snadno vidět, že pro přirozenou implementaci v zásadě stačí paměť velikosti 400 pozic (400 „šachovnic“). (Ano, jedná se o jistý průchod stromem hloubky ≤ 400; přitom není třeba konstruovat v paměti celý strom, ale stačí vždy udržovat aktuální větev.)

Tím jsme si připomněli, že i v malém prostoru (malé paměti) se pochopitelně dají provádět časově náročné výpočty.

Nadefinovali jsme třídy PSPACE, NPSPACE a připomněli jsme si Savitchovu větu z referátu na cvičení (a z učebního textu), která mj. implikuje PSPACE = NPSPACE.

Znázornili jsme si obrázkem inkluze

$$\mathsf{PTIME} \subseteq \mathsf{NPTIME} \subseteq \mathsf{PSPACE} = \mathsf{NPSPACE}.$$

Má se obecně zato, že obě inkluze jsou vlastní, byť nikdo nevyvrátil možnost PTIME = PSPACE.

Připomněli jsme si, co jsou NP-úplné problémy a nadefinovali jsme PSPACE-*úplné problémy*. Jako příklady PSPACE-úplných problému jsme uvedli QBF (problém pravdivosti kvantifikovaných booleovských formulí), Eq-NFA (ekvivalence nedeterministických konečných automatů) a Eq-RegExp (ekvivalence regulárních výrazů). (Žádnému posluchači samozřejmě nedělá nejmenší problém uvést přesné definice problémů a příklady pozitivních a negativních instancí, že ano.)

Uvedli jsme také, že nejrůznější deskové a grafové hry se dají zformalizovat jako PSPACE-těžké (případně PSPACE-úplné) problémy. Např. u šachů by to ovšem chtělo definovat např. $(n \times n)$-šachy (pro všechna n, nejen $n = 8$). (Připomeňme, že podle našich definic patří každý problém s konečně mnoha instancemi do třídy $\mathcal{T}(1)$, tedy má konstantní složitost!) Všimli jsme si, že problém QBF lze definovat jako zjišťování existence vítězné strategie ve hře dvou hráčů, kde Eva („existenční hráč“) nasazuje existenčně vázané proměnné a Adam („univerzální hráč“) nasazuje univerzálně vázané proměnné.

Dokazatelně nezvládnutelné problémy

Jestliže prokážeme NP-obtížnost či PSPACE-obtížnost nějakého problému, říkáme, že je *prakticky nezvládnutelný* (intractable). Je totiž jasné, že navrhneme-li algoritmus, který řeší (přesně ten) daný problém, a neobjevíme-li přitom geniální „trik“, na který dosud nikdo nepřišel, bude mít algoritmus tzv. superpolynomiální (typicky exponenciální či ještě horší) složitost. Obecně používat algoritmus (resp. odpovídající program) budeme moci jen na velmi malá vstupní data. Teoreticky ovšem pořád ještě možnost rychlého algoritmu (založeného na „geniálním triku“) existuje.

Samozřejmě je možné, že exponenciální algoritmus ve skutečnosti chceme použít jen na malá data, anebo ho třeba používáme na data, při nichž se neprojeví ona (worst case) exponenciální složitost. V tomto smyslu praktická nezvládnutelnost (obecného) problému ještě neznamená, že ho v praxi nemůže počítačový program úspěšně řešit v pro nás zajímavých případech. (Existují i jiné možnosti, jak v praxi úspěšně řešit i „nezvládnutelný“ problém. Zmíníme se o tom v partiích o aproximačních a pravděpodobnostních algoritmech.)

Známe ovšem i tzv. *dokazatelně nezvládnutelné* problémy (provably intractable problems), tj. ty, u nichž máme *dokázáno*, že pro ně neexistují polynomiální algoritmy. Definujeme-li např. třídy

$$\mathsf{EXPTIME} = \bigcup_{k=0}^{\infty} \mathcal{T}(2^{n^k}) \,, \qquad \mathsf{EXPSPACE} = \bigcup_{k=0}^{\infty} \mathcal{S}(2^{n^k})$$

pak EXPTIME-těžký či EXPSPACE-těžký problém je takovým dokazatelně nezvládnutelným problémem.

Dá se totiž ukázat, že inkluze $\mathsf{PTIME} \subset \mathsf{EXPTIME}$, $\mathsf{PSPACE} \subset \mathsf{EXPSPACE}$ jsou skutečně vlastní (tzn., že neplatí rovnost). (My to zde ale dokazovat nebudeme.)

Např. následující problém je EXPSPACE-úplný:

NÁZEV: RE^2 (*ekvivalence regulárních výrazů s mocněním*)

VSTUP: dva regulární výrazy, v nichž je možné použít mocnění (tzn. je možno psát α^2 místo $\alpha \cdot \alpha$).

OTÁZKA: reprezentují zadané výrazy tentýž jazyk?

Připomeňme, že problém Eq-RegExp pro standardní regulární výrazy (bez mocnění) je PSPACE-úplný, stejně jako problém Eq-NFA jazykové ekvivalence *nedeterministických* konečných automatů. (Víme, že složitost je funkcí velikosti vstupu; mocnění v regulárních výrazech umožňuje exponenciálně zkrátit zápis některých dlouhých standardních výrazů.)

Existují samozřejmě i dokazatelně těžší (superexponenciální) problémy. Ilustrujme je příkladem problému Presburgerovy aritmetiky (rozhodování pravdivosti formulí teorie sčítání).

Rozhodnutelnost Presburgerovy aritmetiky (jen sčítání)

NÁZEV: ThAdd (*problém pravdivosti teorie sčítání*)

VSTUP: formule jazyka 1. řádu užívající jediný „nelogický“ symbol – ternární (tj. 3-ární) predikátový symbol $PLUS$ ($PLUS(x, y, z) \Leftrightarrow_{df} x + y = z$).

OTÁZKA: je daná formule pravdivá pro množinu $\mathbb{N} = \{0, 1, 2, \dots\}$, kde $PLUS(a, b, c)$ je interpretováno jako $a + b = c$?

Důkaz rozhodnutelnosti Presburgerovy aritmetiky (využívající jen predikát $PLUS(x, y, z)$), tedy důkaz věty 9.3. z kapitoly 9, prodiskutujeme na rozšiřující přednášce.

Nerozhodnutelnost aritmetiky (se sčítáním a násobením)

Uvedli jsme si jen hlavní myšlenku nerozhodnutelnosti pravdivosti uzavřených formulí 1.řádu s predikáty $PLUS(x, y, z)$ (tj. $x + y = z$) a $MULT(x, y, z)$ (tj. $x \cdot y = z$) ve standardním modelu aritmetiky $(\mathbb{N}, +, \cdot)$.

(Využili jsme nerozhodnutelnosti existence akceptujícího výpočtu zadaného Turingova stroje M na zadaném slově w.)

Přednáška - druhá část

Rozhodnutelnost Presburgerovy aritmetiky (jen sčítání).

Zde vůbec není zřejmé, že existuje algoritmus, který daný problém řeší. Presburger ukázal takový algoritmus ve 20. létech 20. století (jedním z cílů tzv. Hilbertova programu bylo ukázat podobný algoritmus i s připuštěním predikátu pro násobení – nemožnost řešení tohoto úkolu ukázal později Gödel). Mnohem později bylo ukázáno, že každý algoritmus, řešící problém ThAdd má složitost minimálně 2^{2^n}.

Presburger ukázal algoritmus využitím tzv. metody eliminace kvantifikátorů. Elegantní důkaz umožňují také výsledky, které známe z teorie konečných automatů.

Věta. Existuje algoritmus rozhodující problém ThAdd.

Představme si třístopou pásku, kde v každé stopě je řetězec nul a jedniček, tj. binární zápis čísla. Na pásku samozřejmě můžeme hledět jako na jednostopou s tím, že povolené *symboly abecedy* jsou uspořádané *trojice* nul a jedniček. Snadno nahlédneme, že existuje konečný automat, který přijímá právě ta slova v „abecedě trojic“, která mají tu vlastnost, že součet čísla v první stopě s číslem v druhé stopě je roven číslu ve třetí stopě. Ihned je to jasné při čtení odzadu; pak si stačí připomenout, že regulární jazyky jsou uzavřeny na zrcadlový obraz.

Z dalších uzávěrových vlastností snadno vyvodíme, že pro libovolnou formuli $\mathcal{F}(x_1, x_2, \dots, x_n)$ jazyka ThAdd která *neobsahuje kvantifikátory*, lze zkonstruovat kon. au-

tomat $A_{\mathcal{F}}$ přijímající právě binární zápisy těch n-tic čísel (na n-stopé pásce), pro které je $\mathcal{F}(x_1, x_2, \ldots, x_n)$ pravdivá.

Pro formuli $(\exists x_n)\mathcal{F}(x_1, x_2, \ldots, x_n)$ je možné zkonstruovat automat, který přijímá právě binární zápisy těch $(n-1)$-tic čísel (dosazených za $x_1, x_2, \ldots, x_{n-1}$), pro které je $(\exists x_n)\mathcal{F}(x_1, x_2, \ldots, x_n)$ pravdivá: automat pracuje na $(n-1)$-stopé pásce, ale simuluje $A_{\mathcal{F}}$ tak, že obsah n-té stopy nedeterministicky hádá! (Pak ho samozřejmě lze převést na ekvivalentní deterministický automat.)

Jelikož $(\forall x_n)\mathcal{F}(x_1, x_2, \ldots, x_n)$ je ekvivalentní $\neg(\exists x_n)\neg\mathcal{F}(x_1, x_2, \ldots, x_n)$, načrtli jsme takto postup, který k formuli jazyka ThAdd v prenexní formě postupnou aplikací zmíněných konstrukcí sestrojí konečný automat, který přijme prázdné slovo (neboli nějakou „posloupnost 0-tic“) právě tehdy, když výchozí formule je pravdivá.

Partie textu k prostudování

Části 8.3., 8.4., 8.5. (třída PTIME, třída NPTIME, NP-úplné problémy). Kapitola 9.

Cvičení

Příklad 13.1

Diskutujte ukázkovou zkouškovou písemku. Speciálně se zaměřte na druhou část, v níž rovněž musíte získat předepsané minimum bodů. (Turingovy stroje, RAMy, polynomiální převeditelnost mezi problémy, konkrétní pozitivní a negativní instance rozhodovacích problémů, aplikace Riceovy věty, základní třídy složitosti a jejich úplné problémy, …)