Týden 12

Přednáška

PSPACE, NPSPACE, PSPACE-úplnost

Uvědomili jsme si, 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 uvedli si Savitchovu větu (z učebního textu), která mj. implikuje PSPACE = NPSPACE.
Uvědomili jsme si 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). (Každý posluchač je jistě

již schopen každý námi zkoumaný problém přesně specifikovat a uvést příklady pozitivních a negativních instancí ...)

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:

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.

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.)

Partie textu k prostudování

Kapitola 9 (speciálně Třída PSPACE).

Cvičení

Příklad 12.1

Definujte třídu PSPACE a pojem „PSPACE-úplný problém“; příkladem je:

NÁZEV: QBF (*problém pravdivosti kvantifikovaných booleovských formulí*)

VSTUP: formule $(\exists x_1)(\forall x_2)(\exists x_3)(\forall x_4)\ldots(\exists x_{2n-1})(\forall x_{2n})\mathcal{F}(x_1, x_2, \ldots, x_{2n})$, kde $\mathcal{F}(x_1, x_2, \ldots, x_{2n})$ je booleovská formule v konjunktivní normální formě.

OTÁZKA: je daná formule pravdivá?

Uveďte nějaké malé, ale netriviální, příklady pozitivních a negativních instancí problému.

(Zbytek příkladu je nepovinný.)

Definujte pravidla hry pro hráče Eva („existenční hráč“) a Adam („univerzální hráč“), kteří postupně nasazují hodnoty proměnných. Jde o to, definovat hru tak, aby Eva měla vítěznou strategii (mimochodem, co to je vítězná strategie?) právě tehdy, když je zadaná

formule pravdivá (a Adam měl vítěznou strategii právě tehdy, když je zadaná formule nepravdivá).
Zbude-li čas, nakonec ilustrujte na malém příkladu, jak lze obecnou plně kvantifikovanou booleovskou formuli ϕ převést (v polynomiálním čase) na ekvivalentní ϕ', která je ve tvaru požadovaném pro vstup problému QBF.

Příklad 12.2

Nejprve objasněte, co to je třída NPSPACE.
Uvažujme nyní nedeterministický Turingův stroj $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, jehož prostorová složitost je omezena polynomem p (při práci na vstupu délky n navštíví jakýkoli výpočet stroje M nejvýš $p(n)$ políček pásky. Nechť c je konstanta zaručující, že všech možných konfigurací uqv stroje M, kde $|uv| \leq \ell$, je nejvýše $2^{c\ell}$ (pro vš. $\ell \geq 0$).
Vysvětlete, co dělá následující funkce (programová procedura):

```
function XY (w: iniciální obsah pásky stroje M)

  function R (n:integer, C,C': konfigurace stroje M, k:integer)
    if k=0 then
      if C = C' or C |- C'
        (* tj.: stroj M může nejvýš jedním krokem přejít z C do C' *)
        then return TRUE else return FALSE
    if k>0 then
      for all C'' of size p(n) do
        if R(n,C,C'',k-1) and R(n,C'',C',k-1)
        then return TRUE
      return FALSE

  for all C of size p(n) do
    if C je koncová (* tedy tvaru uqv, kde q je v F *)
       and R(|w|, q0 w , C , c p(|w|))
    then return TRUE
  return FALSE
```

Odvoďte, jakou paměťovou složitost funkce `XY` má. (Speciálně vezměte v potaz, že ona dvě volání `R(n,C,C'',k-1) and R(n,C'',C',k-1)` mohou pro svou práci využívat tutéž pracovní paměť.)
Je už teď jasné, že PSPACE = NPSPACE ?

Příklad 12.3

Co myslíte, je třída NPSPACE uzavřena na doplněk? (Tedy platí, že pro každý problém v NPSPACE je jeho doplňkový problém [s obrácenou otázkou] také v NPSPACE ?) (Nápověda: využijte faktu, že PSPACE = NPSPACE.)

Příklad 12.4

Uvažujte následující problém.

NÁZEV: Eq-NFA (*problém ekvivalence nedeterministických konečných automatů*)

VSTUP: dva NFA A_1, A_2.

OTÁZKA: platí $L(A_1) = L(A_2)$?

Připomeňte si nejdříve algoritmus rozhodující daný problém, který znáte. Prokázali jste jím, že problém Eq-NFA patří do EXPTIME ?

Zamyslete se, zda se vám podaří ukázat příslušnost problému Eq-NFA k PSPACE. (Nápověda. Jak víme, stačí ukázat že Non-Eq-NFA (doplněk k Eq-NFA) patří do NPSPACE. Vysvětlete, proč to stačí, a pak se to snažte ukázat.)

(Pozn. Eq-NFA je také jeden ze známých PSPACE-úplných problémů.)