VCS – Přednáška 9

1 Paměťová složitost

$$\mathrm{SPACE}(f(n)) = \bigcup_k \{L \mid L \text{ je jazyk rozhodovaný v paměti } \mathcal{O}(f(n))\} \tag{1}$$

$$\mathrm{NSPACE}(f(n)) = \bigcup_k \{L \mid L \text{ je jazyk nedeterministicky rozhodovaný v paměti } \mathcal{O}(f(n))\} \tag{2}$$

Theorem 1 (Savitchova věta). *Nechť $f : \mathbb{N} \to \mathbb{R}$, t.ž. $n \leq f(n)$. Pak platí*

$$\mathrm{SPACE}(f^2(n)) \supseteq \mathrm{NSPACE}(f(n)).$$

Důkaz. Nechť N je NTS rozhodující A v paměti $f(n)$. Zkonstruujeme deterministický TS M, který rozhoduje A. TS M používá proceduru (TS) CanYield, která testuje jestlí jedna z konfiguraci NTS N může odvodit druhou v daném počtu kroků. Pro konfigurace c_1 a c_2 NTS N a číslo $t \in \mathbb{N}$, CanYield(c_1, c_2, t) přijme, pokud c_2 lze odvodit z c_1 v t krocích, jinak zamítne. (Značení $\lceil \cdot \rceil$ představuje zaokrouhlení nahoru).

TS CanYield pro vstupní slovo $[c_1, c_2, t]$:
1. Pokud $t = 1$, testuje jestli $c_1 = c_2$ nebo $c_1 \vdash_N c_2$ pokud ano, přijme, jinak zamítne.
2. Pokud $t > 1$ tak pro každou konfiguraci c_m, která využívá prostor $f(n)$: Spustí CanYield pro $[c_1, c_m, \lceil \frac{t}{2} \rceil]$ a spustí CanYield pro $[c_m, c_2, \lceil \frac{t}{2} \rceil]$, pokud obě spuštění skončí přijetím, přijme.
3. Pokud neexistuje taková konfigurace c_m, pro které by došlo k přijetí v předchozím bodu, zamítne.

Nyní definujeme M tak aby simuloval N následovně: Nejdříve upravíme N tak, aby po sobě uklízel – označme jeho unikátní přijímající konfiguraci $\mathcal{C}^+$. Určíme konstantu d tak aby N nemělo více než $2^{df(n)}$ konfigurací používajících $f(n)$ políček pásky ($n = |w|$). Víme, že $2^{df(n)}$ je horní hranice pro čas kterékoli větve výpočtu N nad w.

TS M pro w:
1. spusť CanYield$(\mathcal{C}_0^w, \mathcal{C}^+, 2^{df(n)})$.

Algoritmus CanYield zjevně řeší problém odvoditelnosti, a tedy N korektně simuluje N. Potřebujeme ho analyzovat, abycho ověřili, jestli M pracuje v paměti $\mathcal{O}(f^2(n))$.

Kdykoli CanYield rekurzivně vyvolá sám sebe, uloží si číslo vnoření, c_1, c_2 a t na zásobník, aby je mohl obnovit při návratu z rekurzivního vyvolání. Každá úrovneň rekurze tedy použije $\mathcal{O}(f(n))$ prostoru navíc. Dále každá úroveň rekurze rozděli velikost t na polovinu. Na začátku je t rovno $2^{df(n)}$, takže hloubka rekurze je $\mathcal{O}(\log 2^{df(n)})$, neboli $\mathcal{O}(f(n))$. A tedy celková použitá paměť je $\mathcal{O}(f^2(n))$.

Ještě jedna technická záležitost vyvstává, protože algoritmus M potřebuje znát hodnotu $f(n)$, když vyvolává CanYield. Toto můžeme vyřešit modifikací M tak, že postupně zkouší hodnoty $f(n) = 1, 2, 3, \ldots$. Pro každou hodnotu $f(n) = i$, ten modifikovaný algoritmus použije CanYield, aby určil N použije alespoň $i+1$ paměti tak, že testuje jestli N může dosáhnout nějaké konfigurace délky $i+1$ ze startovní konfigurace. Pokud je přijímající konfigurace dosažitelná, M přijímá; pokud žádná konfugurace délky $i+1$ není dosažitelná, M zamítá; a jinak M pokračuje hodnotou $f(n) = i + 1$.

2 Třída PSPACE (a třída NPSPACE) a PSPACE-úplné problémy

Definition 1. *Třída* PSPACE *je definována jako*

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}(n^k).$$

Remark 1. Analogicky k PSPACE bychom mohli definovat NPSPACE. Ze Savitchovy věty ale plyne, že by platilo PSPACE = NPSPACE.

Definition 2. *A je* PSPACE*-úplný jazyk, pokud*

- $A \in$ PSPACE
- *pro každý* $B \in$ PSPACE *platí, že* $B \leq_p A$.

Definition 3 (TQBF). *Jazyk* TQBF *je definován jako*

$$\text{TQBF} = \{[\phi] \mid \ \phi \text{ je pravdivá, plně kvantifikovaná Booleovská formule}\}.$$

Theorem 2. *Jazyk* TQBF *je* PSPACE*-úplný.*

Důkaz. Nejdříve ukážeme, že TQBF $\in$ PSPACE

TS T pro $[\phi]$, kde ϕ je plně kvantifikovaná Booleovská formule:

1. Pokud ϕ neobsahuje žádné kvantifikátory, je to výraz obsahující pouze konstaty. Vyhodnoť ϕ, pokud je pravdivá, *přijmi*, jinak *zamítni.*
2. Pokud ϕ je rovna $\exists x\psi$, rekurzivně vyvolej T na ψ, nejprve x nahraď nulami, a poté x nahraď jedničkami. Pokud alespoň jedno vyvolání skončilo přijetím, *přijmi* jinak *zamítni.*
3. Pokud ϕ je rovna $\forall x\psi$, rekurzivně vyvolej T na ψ, nejprve x nahraď nulami, a poté x nahraď jedničkami. Pokud alespoň obě vyvolání skončila přijetím, *přijmi* jinak *zamítni.*

TS T zjevně rozhoduje TQBF. Abychom analyzovali jeho paměťovou složitost, všimněme si, že hloubka rekurze je nejvýše množství proměnných. Na každé úrovni si potřebujeme uložit jenom hodnotu jedné proměnné, žakže celková použitá paměť je $\mathcal{O}(m)$, kde m je počet proměnných, které se vyskytují v ϕ. A tedy T pracuje v linearním čase.

Dále ukážeme že TQBF je PSPACE-těžký. Nechť A je jazyk rozhodovaný TS M v paměti n^k pro nějakou konstantu k. Popíšeme redukci v polynomickém čase z A na TQBF.

Ta redukce zobrazuje w na kvantifikovanou formuli ϕ právě tehdy, když M přijímá w. Abychom zkonstruovali ϕ, budeme řešit obecnější problém. Pokud máme dvě kolekce proměnných, označené c_1 a c_2, které reprezentují dvě konfigurace a číslo t, zkonstruujeme formuli $\phi_{c_1,c_2,t}$. Pokud přiřadíme c_1 a c_2 aktuálním konfiguracím, formula je pravdivá právě tehdy a jen tehdy, pokud se M může dostat z c_1 do c_2 během t kroků. Poté sestavíme ϕ jako formuli $\phi_{c_{\text{start}},c_{\text{accept}},j}$, kde $h = 2^{df(n)}$ pro nějakou konstantu d, vybranou tak, že M nemá více než $2^{df(n)}$ možných konfigurací na vstupu délky n. Pro jednosduchost předpokladejme t je mocnina 2.

Formule bude kódovat obsah políček pásky tak jako v důkazu Cookovy věty. Každé políčko má několik proměnných, jeden pro každý symbol páskové abecedy a stav, odpovídající možným status;m každého políčka. Každá konfigurace má n^k políček a proto je zakódovaná $\mathcal{O}(n^k)$ proměnnými.

Pokud $t = 1$, můžeme jednoduše zkonstruovat $\phi_{c_1,c_2,i}$. Navrhneme tu formuli tak, aby říkala, že c_1 je rovna c_2 nebo že z c_2 lze odvodit c_1 v jednom kroku výpočtu.

Pokud $t > 1$ zkonstruujeme $\phi_{c_1,c_2,t}$, rekurzivně:

$$\phi_{c_1,c_2,t} = \exists m_1 \forall (c_3, c_4) \in \{(c_1, m_1), (m_1, c_2)\} \phi_{c_3,c_4,\frac{t}{2}}.$$

Zápisem $\forall (c_3, c_4) \in \{(c_1, m_1), (m_1, c_2)\}$ indikujeme, že proměnné reprezentující konfigurace c_3 a c_4 mohou nabývat hodnot c_1 a m_1 nebo hodnot m_1 a c_2, a že výsledná formule $\phi_{c_3,c_4,\frac{t}{2}}$ je pravdivá v obou případech. Konstrukci $\forall x \in \{y, z\}[\dots]$ ekvivalentní konstrukcí $\forall x[(x \leftrightarrow y \vee x \leftrightarrow z) \rightarrow \dots]$, abychom dostali syntakticky korektní Booleovskou formuli ($\rightarrow$ a $\leftrightarrow$ lze vyjádřit pomocí *and* a *not*).

Abychom vypočítali velikost formule $\phi_{c_1,c_2,t}$, kde $h = 2^{df(n)}$, všimněme si, že každá úroveň rekurze má velikost $\mathcal{O}(f(n))$. Počet úrovní té rekurze je $\log(2^{df(n)})$, neboli $\mathcal{O}(f(n))$. Takže velikost výsledné formule je $\mathcal{O}(f^2(n))$.

3 Třída L, třída NL a NL-úplné problémy

Potřebujeme upravit výpočetní model: Uvažujeme TS s dvěma páskami:

- *read-only* vstupní páska,
- *read/write* pracovní páska.

A dodáme způsob detekce konce začátku a konce vstupu (např. zarážky jako u LBA), hlava na vstupní pásce musí zůstat v části, na které je zapsán vstup. Pouze políčka použitá na pracovní pásce se počítají do paměťové složitosti tohoto typu TS.

Definition 4. $\mathcal{L}$ *je třída jazyků, které jsou rozhodované v logaritimickém čase na deterministickém Turingově stroji. Neboli,*

$$\mathcal{L} = \text{SPACE}(\log n).$$

$\mathcal{NL}$ *je třída jazyků, které jsou rozhodované v logaritimickém čase na nedeterministickém Turingově stroji. Neboli,*

$$\mathcal{NL} = \text{NSPACE}(\log n).$$

Example 1. (a) $\{0^k 1^k \mid k \geq 0\} \in \mathcal{L}$,

(b) PATH $\in \mathcal{NL}$.

Definition 5. Překladač v logaritimické paměti *je TS s* read-only *vstupní páskou,* write-only *výstupní páskou a* read/write *pracovní páskou. Pracovní páska může obsahovat jen* $\mathcal{O}(\log n)$ *symbolů.*

Překladač v logaritimické paměti M *počítá funkci* $f : \Sigma^* \to \Sigma^*$*, kde* $f(w)$ *je řetězec, který zůstane na výstupní pásce poté, co* M *zastaví, pokud začal s* w *na vstupní pásce.*

Funkci f *nazýváme* funkcí vyčíslitelnou v logaritmické paměti*. Jazyk* A *nazveme jazyk redukovatelný v logaritmické paměti na jazyk* B*, pokud* $A \leq B$ *a redukce je vyčíslitelná v logaritmické paměti (zapisujeme* $A \leq_L B$*).*

Definition 6. *Jazyk* A *je* $\mathcal{NL}$*-úplný pokud*

- $A \in \mathcal{NL}$
- *každý* $B \in \mathcal{NL}$ *je redukovatelný v logaritmické paměti na* B*.*

Theorem 3. *Pokud* $A \leq_L B$ *a* $B \in \mathcal{L}$*, pak* $A \in \mathcal{L}$*.*

Theorem 4. *Jazyk* PATH *je* $\mathcal{NL}$*-úplný.*