

Vlastnosti tříd složitosti

k-páskové Turingovy stroje

Věta 12.1 Je-li jazyk L přijímán nějakým k -páskovým DTS M_k v čase $t(n)$, pak je také přijímán nějakým 1-páskovým DTS M_1 v čase $O(t(n)^2)$.

Důkaz. (idea)

- Obsah k pásek stroje M_k můžeme zapsat na jednu pásku stroje M_1 za sebe a oddělit je vhodným oddělovačem. Pozici hlav M_k na pásce můžeme zakódovat vhodným označením symbolů, pod kterými se hlavy aktuálně nacházejí.
- Při simulaci kroku stroje M_k stroj M_1 dvakrát projde páskou – při jednom průchodu zjistí znaky pod hlavami M_k , při druhém je patřičně upraví.
- Jestliže je na některé simulované pásce stroje M_k nedostatek prostoru, je zapotřebí provést posun zbytku pásky stroje M_1 , což si může opět vyžádat dva průchody touto páskou.
- Užitečná délka pásek stroje M_k je ovšem omezena na $t(n)$ a tudíž užitečná délka pásky stroje M_1 je omezena na $k \cdot t(n) + k$, tj. $O(t(n))$.
- Simulace jednoho kroku M_k strojem M_1 je proto v $O(t(n))$.
- Takových kroků se provede $t(n)$ a tudíž M_1 přijímá L v čase $O(t(n)^2)$.

□

Determinismus a nedeterminismus

Věta 12.2 Je-li jazyk L přijímán nějakým NTS M_n v čase $t(n)$, pak je také přijímán nějakým DTS M_d v čase $2^{O(t(n))}$.

Důkaz. (idea) Ukážeme, jak může M_d simulovat M_n v uvedeném čase:

- Očíslujeme si přechody M_n jako $1, 2, \dots, k$.
- M_d bude postupně simulovat veškeré možné posloupnosti přechodů M_n (obsah vstupní pásky si uloží na pomocnou pásku, aby ho mohl vždy obnovit; na jinou pásku si vygeneruje posloupnost přechodů z $\{1, 2, \dots, k\}^*$ a tu simuluje).
- Vzhledem k možnosti nekonečných výpočtů M_n nelze procházet jeho možné výpočty do hloubky – budeme-li je ale procházet do šířky (tj. nejdřív všechny řetězce z $\{1, 2, \dots, k\}^*$ délky 1, pak 2, pak 3, ...), určitě nalezneme nejkratší přijímající posloupnost přechodů pro M_n , existuje-li.
- Takto projdeme nanejvýš $O(k^{t(n)})$ cest, simulace každé z nich je v $O(t(n))$ a tudíž celkem využijeme nanejvýš čas $O(k^{t(n)})O(t(n)) = 2^{O(t(n))}$.

□

❖ Doposud nikdo nebyl schopen přijít s polynomiální simulací NTS pomocí DTS. **Zdá se tedy, že nedeterminismus přináší značnou výhodu z hlediska časové složitosti výpočtů.**

❖ Zatímco se zdá, že nedeterminismus přináší značnou výhodu z hlediska časové složitosti výpočtů, v případě prostorové složitosti je situace jiná:

Věta 12.3 (Savitchův teorém) $NSpace[s(n)] \subseteq DSpace[s^2(n)]$ pro každou prostorově zkonstruovatelnou funkci $s(n) \geq \lg n$.

Důkaz. Uvažme NTS $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ rozhodující $L(M)$ v prostoru $s(n)$:

- Existuje $k \in \mathbb{N}$ závislé jen na $|Q|$ a $|\Gamma|$ takové, že pro libovolný vstup w , M projde nanejvýš $k^{s(n)}$ konfigurací o délce max. $s(n)$.
- To implikuje, že M provede pro w nanejvýš $k^{s(n)} = 2^{s(n)\lg k}$ kroků.
- Pomocí DTS můžeme snadno implementovat proceduru $test(c, c', i)$, která otestuje, zda je možné v M dojít z konfigurace c do c' v 2^i krocích:

procedure $test(c, c', i)$

if $(i = 0)$ **then return** $((c = c') \vee (c \stackrel{M}{\vdash} c'))$

else for all configurations c'' **such that** $|c''| \leq s(n)$ **do**

if $(test(c, c'', i - 1) \wedge test(c'', c', i - 1))$ **then return true**

return false

Důkaz pokračuje dále.

Pokračování důkazu.

- Všimněme si, že rekurzivním vyvoláváním $test$ vzniká strom o výšce i simulující posloupností svých listů posloupnost 2^i výpočetních kroků.
- Nyní k deterministické simulaci M postačí projít všechny akceptující konfigurace c_F takové, že $|c_F| \leq s(n)$, a ověřit, zda $test(c_o, c_f, \lceil s(n) \lg k \rceil)$, kde c_o je počáteční konfigurace.
- Každé vyvolání $test$ zabere prostor $O(s(n))$, hloubka rekurze je $\lceil s(n) \lg k \rceil = O(s(n))$ a tedy celkově deterministicky simulujeme M v prostoru $O(s^2(n))$.
- Dodejme, že $s(n)$ může být zkonstruováno v prostoru $O(s(n))$ (jedná se o prostorově zkonstruovatelnou funkci) a tudíž neovlivňuje výše uvedené úvahy.

□

❖ Důsledkem Savitchova teorému jsou již dříve uvedené rovnosti:

- **PSPACE** \equiv **NPSPACE**,
- **k-EXSPACE** \equiv **k-NEXSPACE**.

Prostor kontra čas

❖ Intuitivně můžeme říci, že zatímco prostor může růst relativně pomalu, čas může růst výrazně rychleji, neboť můžeme opakovaně procházet týmiž buňkami pásky – opačně tomu být zřejmě nemůže (nemá smysl mít nevyužitý prostor).

Věta 12.4 $*NSpace[t(n)] \subseteq DTime[O(1)^{t(n)}]$ pro každou časově zkonstruovatelnou funkci $t(n) \geq \lg n$.

Důkaz. Dá se použít do jisté míry podobná konstrukce jako u Savitchova teorému – blíže viz literatura. □ *

Věty o kompresi prostoru a zrychlení

Věta 12.5 Necht' M je DTS. Pak existuje ekvivalentní TS N takový, že

$$S_N(n) \leq \frac{S_M(n)}{2} + 2$$

Důkaz. (Idea) Abeceda stroje N obsahuje dvojice znaků abecedy stroje M . Obsah pásky a_1, a_2, \dots, a_n stroje M pak bude reprezentován jako

$$\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}, \begin{pmatrix} a_2 \\ a_3 \end{pmatrix}, \dots$$

Věta 12.6 Necht' M je DTS. Pak existuje ekvivalentní TS N takový, že

$$T_N(n) \leq \frac{T_M(n)}{2} + 2n$$

Důkaz. (Idea) Stroj N si předpočítá výsledky všech možných výpočtů délky 2 stroje M .

Uzavřenost vůči doplňku

❖ **Doplňkem třídy** rozumíme třídu jazyků, které jsou doplňkem jazyků dané třídy. Tedy označíme-li doplněk třídy \mathcal{C} jako $co-\mathcal{C}$, pak $L \in \mathcal{C} \Leftrightarrow \bar{L} \in co-\mathcal{C}$. U rozhodování problémů toto znamená rozhodování komplementárního problému (prázdnota x neprázdnota apod.).

❖ **Prostorové třídy** jsou obvykle uzavřeny vůči doplňku:

Věta 12.7 *Jestliže $s(n) \geq \lg n$, pak $NSpace(s(n)) = co-NSpace(s(n))$.

Důkaz. Jedná se o teorém Immermana a Szelepcsényiho – důkaz viz literatura. □ *

❖ Pro **časové třídy** je situace jiná:

- Některé třídy jako **P** či **EXP** jsou uzavřeny vůči doplňku.
- U jiných významných tříd zůstává otázka uzavřenosti vůči doplňku otevřená. Proto má smysl hovořit např. i o třídách jako:
 - **co-NP** či
 - **co-NEXP**.

Věta 12.8 Třída **P** je uzavřena vůči doplňku.

Důkaz. (idea) Základem je to, že ukážeme, že jestliže jazyk L nad Σ může být přijat DTS M v polynomiálním čase, pak také existuje DTS M' , který L rozhoduje v polynomiálním čase, tj. $L = L(M')$ a existuje $k \in \mathbb{N}$ takové, že pro každé $w \in \Sigma^*$ M' zastaví v čase $O(|w|^k)$:

- M' na začátku výpočtu určí délku vstupu w a vypočte $p(|w|)$, kde $p(n)$ je polynom určující složitost přijímání strojem M . Na speciální dodatečnou pásku uloží $p(|w|)$ symbolů.
- Následně M' simuluje M , přičemž za každý krok umaže jeden symbol z dodatečné pásky. Pokud odebere z této pásky všechny symboly a M by mezitím nepřijal, M' abnormálně ukončí výpočet (odmítne).
- M' evidentně přijme všechny řetězce, které přijme M na to mu stačí $p(n)$ simulovaných kroků a nepřijme všechny řetězce, které by M nepřijal – pokud M nepřijme v $p(n)$ krocích, nepřijme vůbec. M' však vždy zastaví v $O(p(n))$ krocích.

□

Ostrost hierarchie tříd

❖ Z dosavadního můžeme shrnout, že platí následující:

- **LOGSPACE \subseteq NLOGSPACE \subseteq P \subseteq NP**
- **NP \subseteq PSPACE = NPSpace \subseteq EXP \subseteq NEXP**
- **NEXP \subseteq EXPSPACE = NEXPSPACE \subseteq 2-EXP \subseteq 2-NEXP \subseteq ...**
- **... \subseteq ELEMENTARY \subseteq PR \subseteq R \subseteq RE ^a**

❖ Řada otázek ostrosti uvedených vztahů pak zůstává otevřená, nicméně z tzv. **teorému hierarchie** (nebudeme ho zde přesně uvádět, neboť je velmi technický – zájemci ho naleznou v literatuře) plyne, že **exponenciální „mezery“ mezi třídami jsou „ostré“**:

- **LOGSPACE, NLOGSPACE \subset PSPACE,**
- **P \subset EXP,**
- **NP \subset NEXP,**
- **PSPACE \subset NEXPSPACE,**
- **EXP \subset 2-EXP, ...**

^aBez důkazu jsme doplnili, že **ELEMENTARY \subset PR.**

Jazyky \mathcal{C} -těžké a \mathcal{C} -úplné

❖ Až doposud jsme třídy používali jako horní omezení složitosti problémů. Všimněme si nyní omezení dolního – to zavedeme pomocí redukovatelnosti třídy problémů na daný problém.

Definice 12.1 Necht' \mathcal{R} je třída funkcí. Jazyk $L_1 \subseteq \Sigma_1^*$ je \mathcal{R} redukovatelný (přesněji \mathcal{R} many-to-one reducible) na jazyk $L_2 \subseteq \Sigma_2^*$, což zapisujeme $L_1 \leq_{\mathcal{R}}^m L_2$, jestliže existuje funkce f z \mathcal{R} taková, že $\forall w \in \Sigma_1^* : w \in L_1 \Leftrightarrow f(w) \in L_2$.

Definice 12.2 Necht' \mathcal{R} je třída funkcí a \mathcal{C} třída jazyků. Jazyk L_0 je \mathcal{C} -těžký (\mathcal{C} -hard) vzhledem k \mathcal{R} redukovatelnosti, jestliže $\forall L \in \mathcal{C} : L \leq_{\mathcal{R}}^m L_0$.

Definice 12.3 Necht' \mathcal{R} je třída funkcí a \mathcal{C} třída jazyků. Jazyk L_0 nazveme \mathcal{C} -úplný (\mathcal{C} -complete) vzhledem k \mathcal{R} redukovatelnosti, jestliže $L_0 \in \mathcal{C}$ a L_0 je \mathcal{C} -těžký (\mathcal{C} -hard) vzhledem k \mathcal{R} redukovatelnosti.

Nejběžnější typy úplnosti

❖ Uvedme nyní **nejběžněji používané typy úplnosti** – všimněme si, že je použita redukovatelnost dostatečně silná na to, aby bylo možné najít úplné problémy vůči ní a na druhou stranu nebyly příslušné třídy triviálně redukovány na jejich (možné) podtřídy:

- **NP, PSPACE, EXP** úplnost je definována vůči **polynomiální redukovatelnosti** (tj. redukovatelnosti pomocí DTS pracujících v polynomiálním čase),
- **P, NLOGSPACE** úplnost definujeme vůči **redukovatelnosti v deterministickém logaritmickém prostoru**,
- **NEXP** úplnost definujeme vůči **exponenciální redukovatelnosti** (tj. redukovatelnosti pomocí DTS pracujících v exponenciálním čase).

Polynomiální redukce

Definice 12.4 *Polynomiální redukce* jazyka L_1 nad abecedou Σ_1 na jazyk L_2 nad abecedou Σ_2 je funkce $f : \Sigma_1^* \rightarrow \Sigma_2^*$, pro kterou platí:

1. $\forall w \in \Sigma_1^* : w \in L_1 \Leftrightarrow f(w) \in L_2$
2. f je vyčíslitelná DTS v polynomiálním čase.

Existuje-li polynomiální redukce jazyka L_1 na L_2 , říkáme, že L_1 se (*polynomiálně*) *redukuje na* L_2 a píšeme $L_1 \leq_P^m L_2$.

Věta 12.9 Je-li $L_1 \leq_P^m L_2$ a L_2 je ve třídě P , pak L_1 je ve třídě P .

Důkaz. Nechť M_f je Turingův stroj, který provádí *redukci* f jazyka L_1 na L_2 a nechť $p(x)$ je jeho časová složitost. Pro libovolné $w \in L_1$ výpočet $f(w)$ vyžaduje nanejvýš $p(|w|)$ *kroků* a produkuje výstup *maximální délky* $p(|w|) + |w|$.

Nechť M_2 přijímá jazyk L_2 v polynomiálním čase daném polynomem $q(x)$.

Uvažujme Turingův stroj, který vznikne kompozicí $M_f M_2$. Tento stroj přijímá jazyk L_1 tak, že pro každé $w \in L_1$ udělá stroj $M_f M_2$ maximálně $p(|w|) + q(p(|w|) + |w|)$ *kroků*, což je polynomem ve $|w|$ a tedy L_1 leží ve třídě P . □

SAT-problém

Problém splnitelnosti – SAT problém

❖ Necht' $V = \{v_1, v_2, \dots, v_m\}$ je konečná množina Booleovských proměnných (prvotních formulí výrokového počtu). *Literálem* nazveme každou proměnnou v_i nebo její negaci $\overline{v_i}$. *Klausulí* nazveme výrokovou formuli obsahující pouze literály spojené výrokovou spojkou \vee (nebo).

Příklady klausulí: $v_1 \vee \overline{v_2}$, $v_2 \vee v_3$, $\overline{v_1} \vee \overline{v_3} \vee v_2$.

❖ *SAT-problém* lze formulovat takto: Je dána množina proměnných V a množina klausulí nad V . Je tato množina klausulí splnitelná?

❖ Každý konkrétní SAT-problém můžeme *zakódovat jediným řetězcem* takto: Necht' $V = \{v_1, v_2, \dots, v_m\}$, každý *literál* v_i zakódujeme řetězcem délky m , který obsahuje samé 0 s výjimkou i -té pozice, která obsahuje symbol p , jde-li o literál v_i , nebo n , jde-li o literál $\overline{v_i}$. *Klausuli* reprezentujeme seznamem zakódovaných literálů oddělených symbolem $/$. *SAT-problém* bude seznam klausulí uzavřených v aritmetických závorkách.

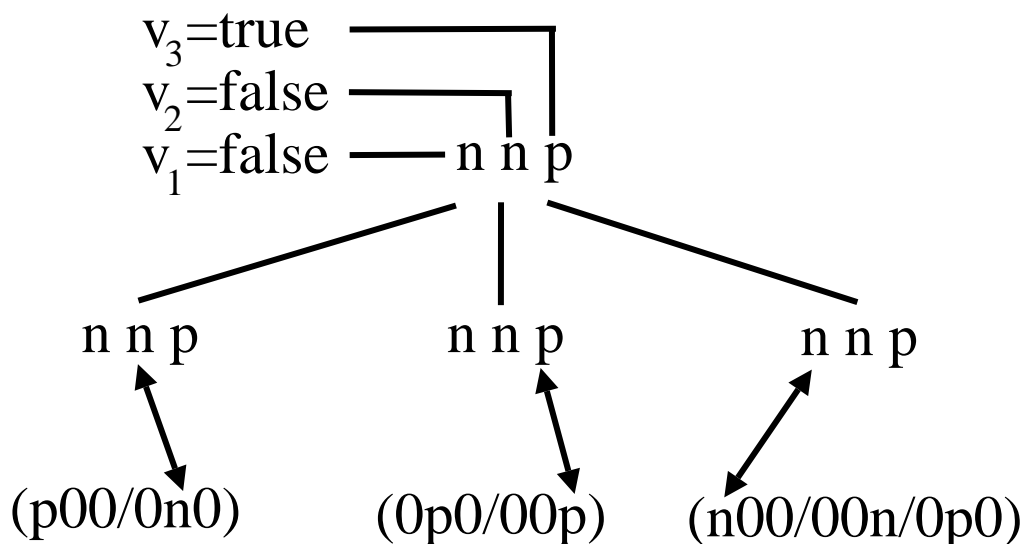
Příklad 12.1 SAT-problém obsahuje proměnné v_1, v_2, v_3 a klausule $v_1 \vee \overline{v_2}$, $v_2 \vee v_3$, $\overline{v_1} \vee \overline{v_3} \vee v_2$ bude reprezentována řetězcem: $(p00/0n0)(0p0/00p)(n00/00n/0p0)$.

❖ Označme L_{SAT} jazyk obsahující řetězce tohoto typu, které reprezentují splnitelné množiny klausulí.

Řetězec $(p00/0n0)(0p0/00p)(n00/00n/0p0)$ je prvkem L_{SAT} ($v_1 = F, v_2 = F, v_3 = T$), na rozdíl od řetězce $(p00/0p0)(n00/0p0)(p00/0n0)(n00/0n0)$, který je kódem nesplnitelné množiny klausulí $v_1 \vee v_2, \overline{v_1} \vee v_2, v_1 \vee \overline{v_2}, \overline{v_1} \vee \overline{v_2}$.

❖ **Přiřazení pravdivostních hodnot** budeme reprezentovat řetězcem z $\{p, n\}^+$, kde p v i -té pozici představuje přiřazení $v_i \approx \text{true}$ a n v i -té pozici představuje přiřazení $v_i \approx \text{false}$.

❖ **Pak test, zda určité hodnocení je modelem množiny klausulí** (množina klausulí je pro toto hodnocení splněna), je velmi jednoduchý a ilustruje ho obrázek:



❖ Na uvedeném principu můžeme zkonstruovat **nedeterministický Turingův stroj**, který přijímá jazyk L_{SAT} v **polynomiálním čase**. Zvolíme 2-páskový Turingův stroj, který:

1. začíná kontrolou, zda vstup reprezentuje množinu klausulí,
2. na 2. pásku nagenereuje řetězec z $\{n, p\}^m$ nedeterministickým způsobem,
3. posouvá hlavu na 1. pásce a testuje, zda pro dané ohodnocení (na 2. pásce) je množina klausulí splnitelná.

Tento proces může být snadno implementován s polynomiální složitostí přijetí v závislosti na délce vstupního řetězce a tedy $L_{SAT} \in NP$.

❖ Princip „guess & check“ použitý výše se často užívá při ukázání **členství v NP**.

Věta 12.10 *Cookův teorém: Je-li L libovolný jazyk z NP , pak $L \leq_P^m L_{SAT}$.*

Hlavní myšlenka důkazu: Protože $L \in NP$, existuje nedeterministický Turingův stroj M a polynom $p(x)$ tak, že pro každé $w \in L$ stroj M přijímá w v maximálně $p(|w|)$ krocích. Jádrem důkazu tvoří konstrukce polynomiální redukce f z L na L_{SAT} : Pro každý řetězec $w \in L$ bude $f(w)$ množina klausulí, které jsou splnitelné, právě když M přijímá w .

NP-úplné jazyky

- ❖ Po objevení Cookova teorému se ukázalo, že mnoho dalších **NP** jazyků má vlastnost podobnou jako L_{SAT} , t.j. jsou polynomiálními redukcemi ostatních **NP** jazyků.
- ❖ Tyto jazyky se – jak již víme – nazývají **NP-úplné** (**NP-complete**) jazyky.
- ❖ Kdyby se ukázalo, že libovolný z těchto jazyků je v **P**, pak by muselo platit **P = NP**; naopak důkaz, že některý z nich leží mimo **P** by znamenalo **P ⊂ NP**.

Význačné NP-úplné problémy

- *Satisfiability*: Je boolovský výraz splnitelný?
- *Clique*: Obsahuje neorientovaný graf kliku velikosti k ?
- *Vertex cover*: Má neorientovaný graf dominantní množinu mohutnosti k ?
- *Hamilton circuit*: Má neorientovaný graf Hamiltonovskou kružnici?
- *Colorability*: Má neorientovaný graf chromatické číslo k ?
- *Directed Hamilton circuit*: Má neorientovaný graf Hamiltonovský cyklus?
- *Set cover*: Je dána třída množin S_1, S_2, \dots, S_n . Existuje podtřída k množin $S_{i_1}, S_{i_2}, \dots, S_{i_k}$ taková, že $\bigcup_{j=1}^k S_{i_j} = \bigcup_{j=1}^n S_j$?
- *Exact cover*: Je dána třída množin S_1, S_2, \dots, S_n . Existuje množinové pokrytí (set cover) tvořené podtřídou po dvojicích disjunktních množin?

Příklad na analýzu složitosti I

Uvažme jazyk

$L_{NE} = \{(\Phi_1, \Phi_2) \mid \Phi_1, \Phi_2 \text{ jsou výrokové formule v konjunktivní normální formě, pro které existuje valuace proměnných } \bar{v} \text{ taková, že } \Phi_1(\bar{v}) \neq \Phi_2(\bar{v})\}.$

Poznámka: $\Phi_i(\bar{v}) \in \{\text{true}, \text{false}\}$ označuje, zda je formule Φ_i pravdivá při valuaci proměnných \bar{v} .

Příklad na analýzu složitosti I

Uvažme jazyk

$L_{NE} = \{(\Phi_1, \Phi_2) \mid \Phi_1, \Phi_2 \text{ jsou výrokové formule v konjunktivní normální formě, pro které existuje valuace proměnných } \bar{v} \text{ taková, že } \Phi_1(\bar{v}) \neq \Phi_2(\bar{v})\}.$

Poznámka: $\Phi_i(\bar{v}) \in \{\text{true}, \text{false}\}$ označuje, zda je formule Φ_i pravdivá při valuaci proměnných \bar{v} .

Nejdříve ukážeme, že $L_{NE} \in \mathbf{EXP}$.

- Sestrojíme DTS M , t.ž. $L(M) = L_{NE}$ a M pracuje v exponenciálním čase.
- M postupně generuje (např. v lexikografickém pořadí) jednotlivé valuace \bar{v} proměnných z formule Φ_1 a Φ_2 .
- Pro každou valuaci \bar{v} M vyhodnotí $\Phi_1(\bar{v})$ a $\Phi_2(\bar{v})$ (lineární průchod oběma formullemi).
- Pokud $\Phi_1(\bar{v}) \neq \Phi_2(\bar{v})$, tak M akceptuje. Pokud prošel všechny valuace a neakceptoval, tak zamítne.
- Pokud Φ_1 a Φ_2 obsahuje n proměnných, pak existuje 2^n různých valuací. Složitost M je tedy v $O(2^n \cdot n) \subseteq O(2^{2n}) \subseteq \mathbf{EXP}$.

Příklad na analýzu složitosti II

Uvažme jazyk

$L_{NE} = \{(\Phi_1, \Phi_2) \mid \Phi_1, \Phi_2 \text{ jsou výrokové formule v konjunktivní normální formě, pro které existuje valuace proměnných } \bar{v} \text{ taková, že } \Phi_1(\bar{v}) \neq \Phi_2(\bar{v})\}.$

Nyní ukážeme, že $L_{NE} \in \mathbf{NP}$.

- Sestrojíme NTS M , t.ž. $L(M) = L_{NE}$ a M pracuje v polynomiálním čase.
- M nedeterministicky zvolí (uhádne) valuaci \bar{v} proměnných z formule Φ_1 a Φ_2 .
- M vyhodnotí $\Phi_1(\bar{v})$ a $\Phi_2(\bar{v})$ (lineární průchod oběma formullemi).
- Pokud $\Phi_1(\bar{v}) \neq \Phi_2(\bar{v})$, tak M akceptuje. V opačném případě zamítne.
- Složitost M je tedy v $O(n)$. Ukázali jsme, že pro L_{NE} existuje polynomiální verifikátor.

Připomeňme, že $L_{NE} \in \mathbf{EXP}$ plyne taktéž přímo z $L_{NE} \in \mathbf{NP}$.

Příklad na analýzu složitosti III

Najděte co nejmenší k takové, aby platilo

- a) $L \in DTIME[n^3] \Rightarrow \{w \in \Sigma^* \mid \exists w_1, w_2, w_3 \in L \text{ t.ž. } w = w_1 w_2 w_3\} \in DTIME[n^k]$
- b) $L \in DTIME[n^3] \Rightarrow \{w \in \Sigma^* \mid \exists w_1, w_2, w_3 \in L \text{ t.ž. } w = w_1 w_2 w_3\} \in NTIME[n^k]$.

Uveďte hlavní myšlenku důkazu (zahrnující konstrukci požadovaného Turingova stroje a jeho časovou analýzu), že pro nalezené k implikace platí. Nedokazujte minimalitu k .

Příklad na analýzu složitosti III

Najděte co nejmenší k takové, aby platilo

- a) $L \in DTIME[n^3] \Rightarrow \{w \in \Sigma^* \mid \exists w_1, w_2, w_3 \in L \text{ t.ž. } w = w_1w_2w_3\} \in DTIME[n^k]$
b) $L \in DTIME[n^3] \Rightarrow \{w \in \Sigma^* \mid \exists w_1, w_2, w_3 \in L \text{ t.ž. } w = w_1w_2w_3\} \in NTIME[n^k]$.

Uveďte hlavní myšlenku důkazu (zahrnující konstrukci požadovaného Turingova stroje a jeho časovou analýzu), že pro nalezené k implikace platí. Nedokazujte minimalitu k .

Označíme $L' = \{w \in \Sigma^* \mid \exists w_1, w_2, w_3 \in L \text{ t.ž. } w = w_1w_2w_3\}$

a) Sestrojíme DTS M' akceptující M' . M' systematicky prochází všechny rozdělení vstupního slova w ($|w| = n$) na podslova w_1, w_2 a w_3 (tj. $w = w_1w_2w_3$). Těchto rozdělení je $O(n^2)$ (potřebujeme umístit 2 "zarážky"). Konstrukce každého rozdělení je v $O(n)$. Pro každé rozdělení zkontrolujeme, zda $w_1 \in L \wedge w_2 \in L \wedge w_3 \in L$, tj. 3-krát voláme DTS M , který akceptuje L v $O(n^3)$. Celkově složitost M' je: $O(n^2) \cdot (O(n) + 3 \cdot O(n^3)) = O(n^5)$. Tudíž $k = 5$.

b) Sestrojíme NTS M' akceptující M' . M' nedeterministicky rozdělí vstupní slovo w na podslova w_1, w_2 a w_3 (tj. $w = w_1w_2w_3$). Toto rozdělení zkonstruuje a ověří požadovanou vlastnost (tj. $w_1 \in L \wedge w_2 \in L \wedge w_3 \in L$) v $O(n) + 3 \cdot O(n^3)$ a tudíž celková složitost M' je v $O(n^3)$. Tudíž $k = 3$.

Příklad na polynomiální redukci

Uvažme jazyk

$L_{NE} = \{(\Phi_1, \Phi_2) \mid \Phi_1, \Phi_2 \text{ jsou výrokové formule v konjunktivní normální formě, pro které existuje valuace proměnných } \bar{v} \text{ taková, že } \Phi_1(\bar{v}) \neq \Phi_2(\bar{v})\}.$

Nyní ukážeme, že L_{NE} je NP-těžký což nám s předchozím důkazem dá NP-úplnost.

Příklad na polynomiální redukci

Uvažme jazyk

$L_{NE} = \{(\Phi_1, \Phi_2) \mid \Phi_1, \Phi_2 \text{ jsou výrokové formule v konjunktivní normální formě, pro které existuje valuace proměnných } \bar{v} \text{ taková, že } \Phi_1(\bar{v}) \neq \Phi_2(\bar{v})\}.$

Nyní ukážeme, že L_{NE} je NP-těžký což nám s předchozím důkazem dá NP-úplnost.

- Ukážeme, že $L_{SAT} \leq_P^m L_{NE}$. Bez ztráty na obecnosti uvažme, že

$L_{SAT} = \{\Phi \mid \Phi \text{ je výroková formule v konjunktivní normální formě, pro kterou existuje valuace proměnných } \bar{v} \text{ taková, že } \Phi(\bar{v}) = \text{true}\}$

- Sestrojíme funkci f (vyčíslitelnou DTS v polynomiálním čase), pro kterou platí $f(\Phi) = (\Phi_1, \Phi_2)$ a $\Phi \in L_{SAT} \iff (\Phi_1, \Phi_2) \in L_{NE}$.
- Stačí položit $\Phi_1 \equiv \Phi$ a $\Phi_2 \equiv x_1 \wedge \bar{x}_1$ (x_1 je proměnná a tudíž Φ_2 je kontradikce).

$$\Phi \in L_{SAT} \Rightarrow \exists \bar{v} : \Phi_1(\bar{v}) = \text{true} \wedge \Phi_2(\bar{v}) = \text{false} \Rightarrow (\Phi_1, \Phi_2) \in L_{NE}$$

$$\Phi \notin L_{SAT} \Rightarrow \forall \bar{v} : \Phi_1(\bar{v}) = \text{false} = \Phi_2(\bar{v}) \Rightarrow (\Phi_1, \Phi_2) \notin L_{NE}$$

Příklady analýzy složitosti mimo Turingovi stroje

Reprezentace grafů

Graf $G = (V, E)$, kde V je množina vrcholů a E je množina hran

- neorientovaný graf: $E = \{\{u, v\} \mid u, v \in V \wedge u \neq v\}$
- orientovaný graf: $E = \{(u, v) \mid u, v \in V\}$

❖ Jak efektivně reprezentovat E

	prostor	dotaz na hranu mezi u, v	vrať následníky u
seznam hran	$O(E)$	$O(E)$	$O(E)$
matice sousednosti	$O(V ^2)$	$O(1)$	$O(V)$
seznam následníků	$O(V + E)$	$O(deg(u)) \leq O(V)$	$O(deg(u)) \leq O(V)$
množina následníků	$O(V + E)$	$O(\log(deg(u)) \leq O(V)$	$O(deg(u)) \leq O(V)$

* $deg(u)$ značí počet následníků vrcholu u (výstupní stupeň).

❖ Poznámka: V případě reprezentace pomocí množiny následníků, složitost dotazu na hranu závisí na datové struktuře použité pro reprezentaci množiny (stromy, hashovací tabulka). Pro jednoduchost budeme předpokládat, že dotaz na hranu umíme v $O(1)$.

Souvislost v neorientovaném grafu

❖ Problém: Je daný graf souvislý: $\forall u, v \in G : \text{cesta}(u, v)$

$\text{cesta}(v_1, v_n) \Leftrightarrow \exists$ posloupnost $v_1 e_1 v_2 e_2 \dots e_{n-1} v_n : \forall 1 \leq i < n : e_i = \{v_i, v_{i+1}\} \wedge e_i \in E$

❖ Naivní algoritmus

$s := \text{randomNode}(G)$

$reach := \{s\}, new := true$

while new **do**

$new := false$

foreach $\{u, v\} \in E$ **do**

if $|\{u, v\} \cap reach| = 1$ **then**

$reach := reach \cup \{u, v\}$

$new := true$

return $V = reach$

Souvislost v neorientovaném grafu

❖ Naivní algoritmus

$s := \text{randomNode}(G)$

$reach := \{s\}, new := true$

while new **do**

$new := false$

foreach $\{u, v\} \in E$ **do**

if $|\{u, v\} \cap reach| = 1$ **then**

$reach := reach \cup \{u, v\}$

$new := true$

return $V = reach$

❖ Složitost: $O(|V| \cdot |E|)$

Souvislost v neorientovaném grafu

❖ Asymptoticky optimální algoritmus

```
foreach  $v \in V$  do  $v.visited := false$   
 $s := \text{randomNode}(G)$ ,  $s.visited := true$ ,  $reach := \{s\}$   
 $Q.enqueue(s)$   
while  $Q.notEmpty()$  do  
     $v := Q.dequeue()$   
    foreach  $u \in Adj(v)$  do  
        if  $u.visited = false$  then  
             $u.visited := true$ ,  $reach := reach \cup \{u\}$   
             $Q.enqueue(u)$   
return  $V = reach$ 
```

Souvislost v neorientovaném grafu

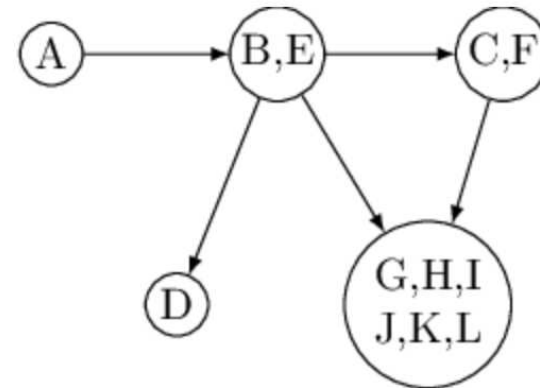
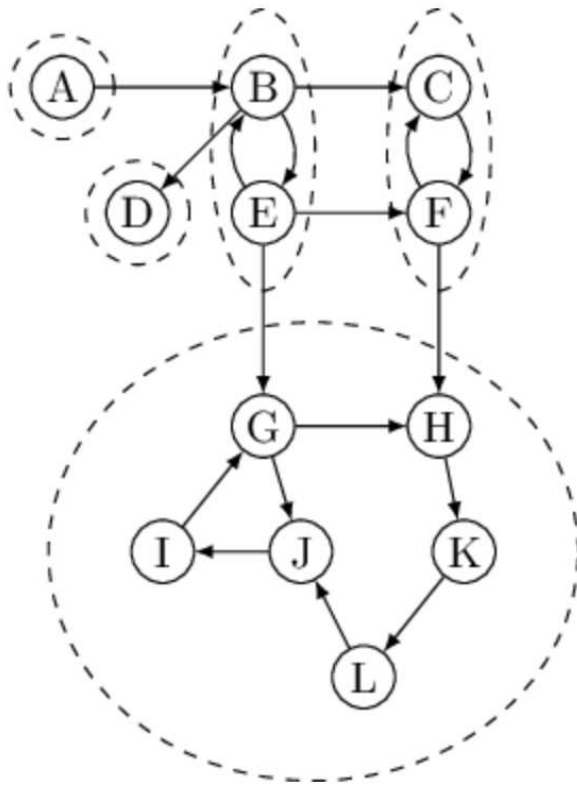
❖ Asymptoticky optimální algoritmus

```
foreach  $v \in V$  do  $v.visited := false$   
 $s := \text{randomNode}(G)$ ,  $s.visited := true$ ,  $reach := \{s\}$   
 $Q.enqueue(s)$   
while  $Q.notEmpty()$  do  
     $v := Q.dequeue()$   
    foreach  $u \in Adj(v)$  do  
        if  $u.visited = false$  then  
             $u.visited := true$ ,  $reach := reach \cup \{u\}$   
             $Q.enqueue(u)$   
return  $V = reach$ 
```

❖ Složitost: $O(|V| + |E|)$

Rozklad na silně souvislé komponenty

- ❖ Rozklad orientovaného grafu na silně souvislé komponenty



Rozklad na silně souvislé komponenty

❖ Naivní algoritmus

```
SCC := ∅  
while  $|V| \neq 0$  do  
   $s := \text{randomNode}(G)$   
   $FWD := \text{Reach}(s, G)$  // states reachable from  $s$  including  $s$   
   $BWD := \text{Reach}(s, G^T)$  // on transposed  $G$  (backward edges)  
   $C := FWD \cap BWD$   
   $SCC := SCC \cup \{C\}$   
   $V := V \setminus C$   
return SCC
```

Rozklad na silně souvislé komponenty

❖ Naivní algoritmus

```
SCC := ∅  
while |V| ≠ 0 do  
  | s := randomNode(G)  
  | FWD := Reach(s, G) // states reachable from s including s  
  | BWD := Reach(s, GT) // on transposed G (backward edges)  
  | C := FWD ∩ BWD  
  | SCC := SCC ∪ {C}  
  | V := V \ C  
return SCC
```

❖ Složitost: $O((|V| + |E|) \cdot |V|)$

Rozklad na silně souvislé komponenty

❖ Tarjanův algoritmus – modifikace DFS

❖ Začneme s DFS, který pro každý vrchol počítá čas návštěvy ($v.time$)

Function Main(G)

foreach $v \in V$ **do** $v.time = -1$

$gTime := 0$

foreach $v \in V$ **do**

if $v.time = -1$ **then**

$gTime := gTime + 1$

$v.time := gTime$

 DFS(v)

Function DFS($v \in V$)

foreach $u \in Adj(v)$ **do**

if $u.time := -1$ **then**

$gTime := gTime + 1$

$u.time := gTime$

 DFS(u)

❖ Složitost: $O(|V| + |E|)$

Tarjanův algoritmus

- ❖ Budeme počítat $v.low$ and udržovat zásobník vrcholů, které nejsou v SCC.

$v.low = \min\{u.time \mid \text{Reach}(v, u) \text{ a } u \text{ byl objeven během volání DFS}(v)\}$

Function Main(G)

```
  foreach  $v \in V$  do
    |  $v.time = -1$ 
     $gTime := 0$ 
    foreach  $v \in V$  do
      | if  $v.time = -1$  then
        |  $gTime := gTime + 1$ 
        |  $v.time := gTime$ 
        |  $v.low := gTime$ 
        |  $stack.push(v)$ 
        | DFS( $v$ )
    return  $SCCs$ 
```

Function DFS($v \in V$)

```
  foreach  $u \in Adj(v)$  do
    | if  $u.time = -1$  then
      |  $gTime := gTime + 1$ 
      |  $u.time := u.low := gTime$ 
      |  $stack.push(u)$ 
      | DFS( $u$ )
      |  $v.low := \min\{v.low, u.low\}$ 
    | else if  $u \in stack$  then
      |  $v.low = \min\{v.low, u.time\}$ 
  if  $v.time = v.low$  then pop the  $stack$ 
  down to  $v$  to create new SCC
```

Tarjanův algoritmus

- ❖ Budeme počítat $v.low$ and udržovat zásobník vrcholů, které nejsou v SCC.

$$v.low = \min\{u.time \mid \text{Reach}(v, u) \text{ a } u \text{ byl objeven během volání DFS}(v)\}$$

Function Main(G)

```
foreach  $v \in V$  do
  |  $v.time = -1$ 
   $gTime := 0$ 
  foreach  $v \in V$  do
    | if  $v.time = -1$  then
      |  $gTime := gTime + 1$ 
      |  $v.time := gTime$ 
      |  $v.low := gTime$ 
      |  $stack.push(v)$ 
      |  $DFS(v)$ 
  return  $SCCs$ 
```

Function DFS($v \in V$)

```
foreach  $u \in Adj(v)$  do
  | if  $u.time = -1$  then
    |  $gTime := gTime + 1$ 
    |  $u.time := u.low := gTime$ 
    |  $stack.push(u)$ 
    |  $DFS(u)$ 
    |  $v.low := \min\{v.low, u.low\}$ 
  | else if  $u \in stack$  then
    |  $v.low = \min\{v.low, u.time\}$ 
if  $v.time = v.low$  then pop the  $stack$ 
down to  $v$  to create new SCC
```

- ❖ Složitost: $O(|V| + |E|)$

Příslušnost do bezkontextového jazyka

- ❖ Uvažme bezkontextovou gramatiku $G = (N, \Sigma, P, S)$ v CNF a slovo w .
- ❖ Platí, že $w \in L(G)$?
- ❖ Připomeňme, že pokud $w \in L(G)$ pak $S \Rightarrow_G^p w \wedge p = 2|w| - 1$ (délka odvození je daná)

Příslušnost do bezkontextového jazyka

- ❖ Uvažme bezkontextovou gramatiku $G = (N, \Sigma, P, S)$ v CNF a slovo w .
- ❖ Platí, že $w \in L(G)$?
- ❖ Připomeňme, že pokud $w \in L(G)$ pak $S \Rightarrow_G^p w \wedge p = 2|w| - 1$ (délka odvození je daná)

Příslušnost do bezkontextového jazyka

- ❖ Uvažme bezkontextovou gramatiku $G = (N, \Sigma, P, S)$ v CNF a slovo w .
- ❖ Platí, že $w \in L(G)$?
- ❖ Připomeňme, že pokud $w \in L(G)$ pak $S \Rightarrow_G^p w \wedge p = 2|w| - 1$ (délka odvození je daná)

```
foreach derivation tree  $T$  in  $G$  of the depth  $p$  do
  | if leafs of  $T$  form  $w$  then
  | | return true
return false
```

Příslušnost do bezkontextového jazyka

- ❖ Uvažme bezkontextovou gramatiku $G = (N, \Sigma, P, S)$ v CNF a slovo w .
- ❖ Platí, že $w \in L(G)$?
- ❖ Připomeňme, že pokud $w \in L(G)$ pak $S \Rightarrow_G^p w \wedge p = 2|w| - 1$ (délka odvození je daná)

```
foreach derivation tree  $T$  in  $G$  of the depth  $p$  do
|   if leafs of  $T$  form  $w$  then
|   |   return true
return false
```

- ❖ Složitost: $2^{O(|w|)}$ algoritmus projde všechny odvození s délkou $O(|w|)$

- připomeňme

$$2^{O(f(n))} = \{g(n) \in \mathcal{F} \mid \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N} \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow g(n) \leq 2^{c \cdot f(n)}\}$$

- pro jednoduchost zanedbáváme $|P|$

Příslušnost do bezkontextového jazyka

❖ Cocke-Younger-Kasami (CYK) algoritmus (dynamické programování)

❖ Hlavní myšlenka: pro každé neprázdné podslovo u slova w (začíná na indexu i a má délku j) spočítáme množinu všech neterminálů z kterých lze u odvodit

$$T_{i,j} = \{X \in N \mid S \Rightarrow_G^* w_i w_{i+1} \dots w_{i+j-1}\}$$

❖ Příklad

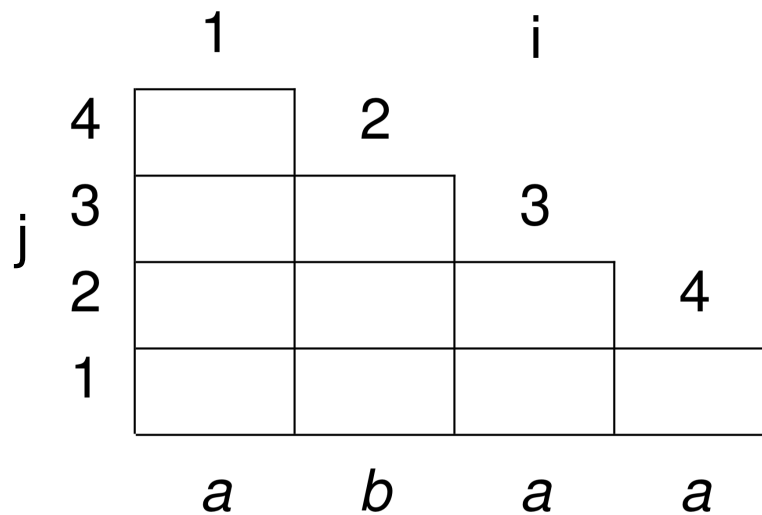
$S \rightarrow AB \mid SS \mid a$

$A \rightarrow AA \mid BC \mid a$

$B \rightarrow AB \mid b$

$C \rightarrow SA \mid b$

$w = abaa$



Příslušnost do bezkontextového jazyka

❖ Cocke-Younger-Kasami (CYK) algoritmus (dynamické programování)

❖ Hlavní myšlenka: pro každé neprázdné podslovo u slova w (začíná na indexu i a má délku j) spočítáme množinu všech neterminálů z kterých lze u odvodit

$$T_{i,j} = \{X \in N \mid S \Rightarrow_G^* w_i w_{i+1} \dots w_{i+j-1}\}$$

❖ Příklad

$S \rightarrow AB \mid SS \mid a$

$A \rightarrow AA \mid BC \mid a$

$B \rightarrow AB \mid b$

$C \rightarrow SA \mid b$

$w = abaa$

❖ $w \in L(G)$ jelikož $S \in T_{1,4}$

		1		i	
	4	S,C,A	2		
	3	S,C	A	3	
j	2	S,B	\emptyset	S,C,A	4
	1	S,A	B,C	S,A	S,A
		a	b	a	a

Cocke-Younger-Kasami algoritmus

Vstup: gramatika $\mathcal{G} = (N, \Sigma, P, S)$ v CNF, slovo $w = w_1 \dots w_n \neq \varepsilon$

Výstup: množiny $T_{i,j} = \{X \in N \mid X \Rightarrow^* w_i \dots w_{i+j-1}\}$

for $i \leftarrow 1$ **to** n **do**

$T_{i,1} \leftarrow \emptyset$

for každé pravidlo tvaru $(A \rightarrow a) \in P$ **do**

if $a = w_i$ **then** $T_{i,1} \leftarrow T_{i,1} \cup \{A\}$

od

od

for $j \leftarrow 2$ **to** n **do**

for $i \leftarrow 1$ **to** $n - j + 1$ **do**

$T_{i,j} \leftarrow \emptyset$

for $k \leftarrow 1$ **to** $j - 1$ **do**

for každé pravidlo tvaru $(A \rightarrow BC) \in P$ **do**

if $B \in T_{i,k} \wedge C \in T_{i+k,j-k}$ **then** $T_{i,j} \leftarrow T_{i,j} \cup \{A\}$

od

od

od

od

Cocke-Younger-Kasami algoritmus

Vstup: gramatika $\mathcal{G} = (N, \Sigma, P, S)$ v CNF, slovo $w = w_1 \dots w_n \neq \varepsilon$

Výstup: množiny $T_{i,j} = \{X \in N \mid X \Rightarrow^* w_i \dots w_{i+j-1}\}$

for $i \leftarrow 1$ **to** n **do**

$T_{i,1} \leftarrow \emptyset$

for každé pravidlo tvaru $(A \rightarrow a) \in P$ **do**

if $a = w_i$ **then** $T_{i,1} \leftarrow T_{i,1} \cup \{A\}$

od

od

for $j \leftarrow 2$ **to** n **do**

for $i \leftarrow 1$ **to** $n - j + 1$ **do**

$T_{i,j} \leftarrow \emptyset$

for $k \leftarrow 1$ **to** $j - 1$ **do**

for každé pravidlo tvaru $(A \rightarrow BC) \in P$ **do**

if $B \in T_{i,k} \wedge C \in T_{i+k,j-k}$ **then** $T_{i,j} \leftarrow T_{i,j} \cup \{A\}$

od

od

od

od

❖ Složitost: $O(|w|^3)$

- opět zanedbáváme $|P|$ – jinak $O(|w|^3 \cdot |P|)$

Univerzalita NKA

- ❖ Pro daný nedeterministický konečný automat $A = \{Q, \Sigma, \delta, s_0, F\}$ rozhodni zda

$$L(A) \neq \Sigma^*.$$

- ❖ Naivní algoritmus

```
Construct deterministic finite automaton  $A'$  such that  $L(A) = L(A')$ ;  
if exists in  $A'$  a path from  $\{s_0\}$  to  $S$  where  $F \cap S = \emptyset$  then  
| return true  
else  
| return false
```

Univerzalita NKA

- ❖ Pro daný nedeterministický konečný automat $A = \{Q, \Sigma, \delta, s_0, F\}$ rozhodni zda

$$L(A) \neq \Sigma^*.$$

- ❖ Naivní algoritmus

```
Construct deterministic finite automataon  $A'$  such that  $L(A) = L(A')$ ;  
if exists in  $A'$  a path from  $\{s_0\}$  to  $S$  where  $F \cap S = \emptyset$  then  
| return true  
else  
| return false
```

- ❖ Prostorová složitost: $2^{O(|Q|)}$

- deterministický automat může mít až exponenciální počet stavů
- dále si musíme uložit přechodovou relaci

- ❖ Časová složitost: $2^{O(|Q|)}$

- hledání neakceptující cesty v exponenciálně velkém automatu

Univerzalita NKA

- ❖ Pro daný NKA $A = \{Q, \Sigma, \delta, s_0, F\}$ rozhodni zda $L(A) \neq \Sigma^*$.
- ❖ Algoritmus pracující v nedeterministickém polynomiálním prostoru
 - on-the-fly determinizace
 - uhodnutí neakceptující cesty

counter := 0;

makroState := $\{s_0\}$;

while *counter* $\leq 2^{|Q|}$ **do**

if *makroState* $\cap F = \emptyset$ **then**
 | **return** *true*

a := nondeterministically choose from Σ ;

makroState := $\bigcup_{q \in \text{makroState}} \delta(q, a)$;

counter := *counter* + 1;

return *false*

Univerzalita NKA

- ❖ Pro daný NKA $A = \{Q, \Sigma, \delta, s_0, F\}$ rozhodni zda $L(A) \neq \Sigma^*$.
- ❖ Algoritmus pracující v nedeterministickém polynomiálním prostoru
 - on-the-fly determinizace
 - uhodnutí neakceptující cesty

counter := 0;

makroState := { s_0 };

while *counter* $\leq 2^{|Q|}$ **do**

if *makroState* $\cap F = \emptyset$ **then**
 | **return** *true*

a := nondeterministically choose from Σ ;

makroState := $\bigcup_{q \in \text{makroState}} \delta(q, a)$;

counter := *counter* + 1;

return *false*

- ❖ Prostorová složitost: $O(|Q|) \subseteq \mathbf{NPSPACE} = \mathbf{PSPACE}$

- pamatujeme si jen 2 makrostavy (každý max $|Q|$) a *counter* ($\log(2^{|Q|}) \in O(|Q|)$)

Univerzalita NKA

- ❖ Pro daný NKA $A = \{Q, \Sigma, \delta, s_0, F\}$ rozhodni zda $L(A) \neq \Sigma^*$.
- ❖ Algoritmus pracující v nedeterministickém polynomiálním prostoru
 - on-the-fly determinizace
 - uhodnutí neakceptující cesty

counter := 0;

makroState := { s_0 };

while *counter* $\leq 2^{|Q|}$ **do**

if *makroState* $\cap F = \emptyset$ **then**
 | **return** *true*

a := nondeterministically choose from Σ ;

makroState := $\bigcup_{q \in \text{makroState}} \delta(q, a)$;

counter := *counter* + 1;

return *false*

- ❖ Časová složitost stále v $2^{O(|Q|)}$
 - pozor není v **NP**, jelikož délka cesty může být exponenciální k $|Q|$ – její ověření není v **P**

Příklady složitosti problémů

❖ Příklady **LOGSPACE** problémů:

- existence **cesty** mezi dvěma uzly v **neorientovaném grafu**.

❖ Příklady **NLOGSPACE-úplných** problémů:

- existence **cesty** mezi dvěma uzly v **orientovaném grafu**,
- **2-SAT** (*SATisfiability*), tj. splnitelnost výrokových formulí tvaru konjunkce disjunkcí dvou literálů (literál je výroková proměnná nebo její negace), např.
 $(x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_3)$.

❖ Příklady **P-úplných** problémů:

- **splnitelnost konjunkce Hornových klauzulí** $(p \wedge q \wedge \dots \wedge t) \Rightarrow u$, kde p, q, \dots jsou atomické formule výrokové logiky (výrokové proměnné),
 - nerozhodnutelné v predikátové podobě
- **náležitost řetězce** do jazyka **bezkontextové gramatiky**: algoritmus „CYK“ založený na dynamickém programování (Cocke, Younger, Kasami) – $O(n^3)$,
- **topologické uspořádání** – pořadí uzlů při průchodu grafem do hloubky (pro dané řazení přímých následníků).

❖ Příklady **NP-úplných** problémů:

- splnitelnost Booleovských formulí (**SAT**, **3SAT**)
- řada grafových a množinových problémů (viz výše)
- **problém obchodního cestujícího**,
- „**knapsack**“ – máme položky s váhou a hodnotou, maximalizujeme hodnotu tak, aby váha nepřekročila určitou mez.

❖ Příklady **co-NP-úplných** problémů:

- **ekvivalence regulárních výrazů bez iterace**.

❖ Příklady **PSPACE**-úplných problémů:

- ekvivalence regulárních výrazů,
- náležitost řetězce do jazyka kontextové gramatiky,
- inkluze jazyků nedeterministických konečných automatů,
- model checking formulí lineární temporální logiky (LTL – výroková logika doplněná o temporální operátory *until*, *always*, *eventually*, *next-time*) vůči velikosti formule.

❖ Příklady **EXP**-úplných problémů:

- inkluze jazyka bezkontextové gramatiky v jazyce NKA (opačná \subseteq nerozh.),
- inkluze nedeterministických konečných *stromových automatů*, zobecňujících přechody KA do podoby $p \xrightarrow{a} (q_1, \dots, q_n)$,
- inkluze pro tzv. *visibly push-down* jazyky (operace push/pop, které provádí přijímající automat, jsou součástí vstupního řetězce),
- nejlepší tah v šachu (zobecněno na šachovnici $n \times n$),
- model checking procesů s neomezeným zásobníkem (rekurzí) vůči zafixované formuli logiky větvícího se času (CTL) – tj. EXP ve velikosti procesu.

❖ Příklady **EXPSPACE**-úplných problémů:

- ekvivalence regulárních výrazů doplněných o operaci kvadrát (tj. r^2).

❖ **k-EXP / k-EXPSPACE:**

- rozhodování splnitelnosti formulí **Presburgerovy aritmetiky** – tj. celočíselné aritmetiky se sčítáním, porovnáváním (ne násobením – to vede na tzv. Peanovu aritmetiku, která je již nerozhodnutelná) a kvantifikací prvního řádu (např. $\forall x, y : x \leq x + y$) je problém, který je v **3-EXP (2-EXPSPACE-úplný)**.

❖ Problémy mimo **ELEMENTARY:**

- ekvivalence regulárních výrazů doplněných o negaci,
- rozhodování splnitelnosti formulí logiky **WS1S** – celočíselná aritmetika s operací +1 a kvantifikací prvního a druhého řádu (tj. pro každou/existuje hodnota, resp. množina hodnot, taková, že ... – např. $\exists A : \forall x \in A : x + 1 \notin A$),
- verifikace dosažitelnosti v tzv. *Lossy Channel Systems* – procesy komunikující přes neomezenou, ale ztrátovou frontu (cokoliv se může kdykoliv ztratit).

Prvočíselný rozklad

- ❖ Problém rozkladu daného celého čísla na součin prvočísel.
- ❖ Velmi důležitý problém pro kryptografii.
- ❖ Ví se, že je v $\text{NP} \cap \text{co-NP}$.
- ❖ Neví se, zda je v P , ani zda je NP -úplný.
- ❖ Existuje polynomiální algoritmus pro test je dané číslo prvočíslo.
- ❖ Existuje polynomiální algoritmus pro kvantové počítače, který počítá rozklad.