

Složitost

Složitost algoritmů

- ❖ Základní teoretický přístup vychází z Church-Turingovy teze:

Každý algoritmus je implementovatelný jistým TS.

- ❖ Zavedení TS nám umožňuje klasifikovat problémy (resp. funkce) do dvou tříd:
 1. problémy, jež nejsou **algoritmicky ani částečně rozhodnutelné** (resp. funkce algoritmicky nevyčíslitelné) a
 2. problémy **algoritmicky alespoň částečně rozhodnutelné** (resp. funkce algoritmicky vyčíslitelné).
- ❖ Nyní se budeme zabývat třídou algoritmicky (částečně) rozhodnutelných problémů (vyčíslitelných funkcí) v souvislosti s otázkou **složitosti** jejich rozhodování (vyčíslování).
- ❖ Analýzu složitosti algoritmu budeme chápat jako analýzu složitosti výpočtů příslušného TS, jejímž cílem je **vyjádřit (kvantifikovat) požadované zdroje (čas, prostor) jako funkci závisující na délce vstupního řetězce**.

Různé případy při analýze složitosti

❖ Můžeme rozlišit:

1. analýzu složitosti nejhoršího případu,
2. analýzu složitosti nejlepšího případu,
3. analýzu složitosti průměrného případu,
4. amortizovanou analýzu

❖ Průměrná složitost algoritmu je definována následovně:

- Jestliže algoritmus (TS) vede k m různým výpočtům (případům) se složitostí c_1, c_2, \dots, c_m , jež nastávají s pravděpodobnostmi p_1, p_2, \dots, p_m , pak **průměrná složitost algoritmu** je dána jako $\sum_{i=1}^m p_i c_i$.

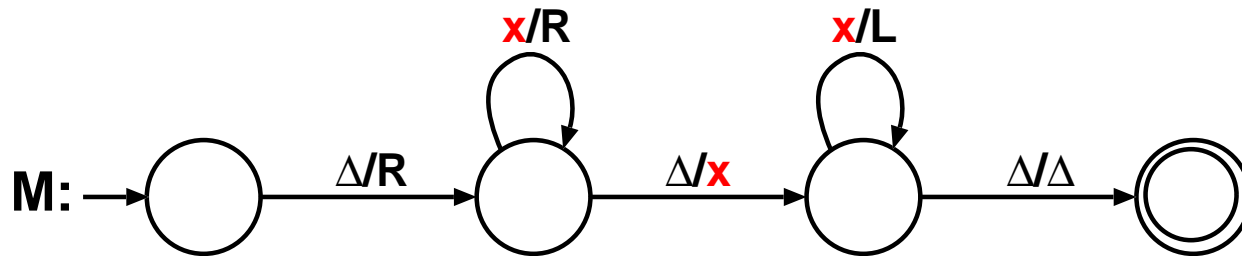
❖ **Amortizovaná analýza** studuje posloupnost operací jako celek. Tato technika umožňuje, na rozdíl od klasického přístupu mnohem přesnější určení časové složitosti algoritmu.

❖ Obvykle (alespoň na teoretické úrovni) se věnuje **největší pozornost** složitosti nejhoršího případu.

Složitost výpočtů TS

- ❖ Časová složitost – počet kroků (přechodů) TS provedený od počátku do konce výpočtu.
- ❖ Prostorová (paměťová) složitost – počet „buněk“ pásky TS požadovaný pro daný výpočet.

Příklad 12.1 Uvažme následující TS M :



Pro vstup $\Delta xxx \Delta \Delta \dots$ je:

- časová složitost výpočtu M rovna 10,
- prostorová složitost výpočtu M rovna 5.

Lemma 12.1 Je-li časová složitost výpočtu prováděného TS rovna n , pak prostorová složitost tohoto výpočtu není větší než $n + 1$.

Důkaz. Tvrzení je jednoduchou implikací plynoucí z definice časové a prostorové složitosti. □

Definice 12.1 Řekneme, že k -páskový DTS (resp. NTS) M přijímá jazyk L nad abecedou Σ v čase $T_M : \mathbb{N} \rightarrow \mathbb{N}$, jestliže $L = L(M)$ a M přijme (resp. může přijmout) každé $w \in L$ v nanejvýš $T_M(|w|)$ krocích.

Definice 12.2 Řekneme, že k -páskový DTS (resp. NTS) M přijímá jazyk L nad abecedou Σ v prostoru $S_M : \mathbb{N} \rightarrow \mathbb{N}$, jestliže $L = L(M)$ a M přijme (resp. může přijmout) každé $w \in L$ při použití nanejvýš $S_M(|w|)$ buněk pásky – nepočítáme zde buňky pásky, na nichž je zapsán vstup, ale nikdy na nich nespočine hlava stroje.

❖ Zcela analogicky můžeme definovat vyčíslování určité funkce daným TS v určitém čase, resp. prostoru.

Analýza složitosti mimo prostředí TS

- ❖ V jiném výpočetním prostředí, než jsou TS, nemusí mít každá primitivní operace stejnou cenu.
- ❖ Velmi často se užívá tzv. **uniformní cenové kritérium**, kdy každé operaci přiřadíme stejnou cenu.
- ❖ Používá se ale např. také tzv. **logaritmické cenové kritérium**, kdy operaci manipulující operand o velikosti i , $i > 0$, přiřadíme cenu $\lfloor \lg i \rfloor + 1$:
 - Zohledňujeme to, že s rostoucí velikostí operandů roste cena operací – logaritmus odráží růst velikosti s ohledem na binární kódování (v n bitech zakódujeme 2^n hodnot).
- ❖ Analýza složitosti za takových předpokladů není zcela přesná, důležité ale obvykle je to, aby se **zachovala informace o tom, jak rychle roste čas/prostor potřebný k výpočtu v závislosti na velikosti vstupu.**^a

^aAlgoritmus A_1 , jehož nároky rostou pomaleji než u jiného algoritmu A_2 , nemusí být výhodnější než A_2 pro řešení malých instancí.

❖ Význam srovnávání rychlosti růstu složitosti výpočtů si snad nejlépe ilustrujeme na příkladu:

Příklad 12.2 Srovnání polynomiální (přesněji kvadratické – n^2) a exponenciální (2^n) časové složitosti:

délka vstupu n	časová složitost $c_1 \cdot n^2, c_1 = 10^{-6}$	časová složitost $c_2 \cdot 2^n, c_2 \doteq 9.766 \cdot 10^{-8}$
10	0.0001 s	0.0001 s
20	0.0004 s	0.1024 s
30	0.0009 s	1.75 min
40	0.0016 s	1.24 dne
50	0.0025 s	3.48 roku
60	0.0036 s	35.68 století
70	0.0049 s	3.65 mil. roků

Složitost výpočtů na TS a v jiných prostředích

❖ Pro rozumná cenová kritéria se ukazuje, že výpočetní složitost je pro různé modely výpočtu blízké běžným počítačům (RAM, RASP stroje) **polynomiálně vázaná** se složitostí výpočtu na TS (viz dále) a tudíž **složitost výpočtu na TS není „příliš“ rozdílná oproti výpočtům na běžných počítačích.**

- **RAM stroje** mají paměť s náhodným přístupem (jedna buňka obsahuje libovolné přirozené číslo), instrukce typu LOAD, STORE, ADD, SUB, MULT, DIV (akumulátor a konstanta/přímá adresa/nepřímá adresa), vstup/výstup, nepodmíněný skok a podmíněný skok (test akumulátoru na nulu), HALT. U RAM stroje je program součástí řízení stroje (okamžitě dostupný), u **RASP** je uložen v paměti stejně jako operandy.
- Funkce $f_1(n), f_2(n) : \mathbb{N} \rightarrow \mathbb{N}$ jsou **polynomiálně vázané**, existují-li polynomy $p_1(x)$ a $p_2(x)$ takové, že pro všechny hodnoty n je $f_1(n) \leq p_1(f_2(n))$ a $f_2(n) \leq p_2(f_1(n))$.
- **Logaritmické cenové kritérium** se uplatní, uvažujeme-li možnost násobení dvou čísel. Jednoduchým cyklem typu $A_{i+1} = A_i * A_i$ ($A_0 = 2$) jsme schopni počítat 2^{2^n} , což TS s omezenou abecedou není schopen provést v polynomiálním čase (pro uložení/načtení potřebuje projít 2^n buněk při použití binárního kódování).

❖ Ilustrujme si nyní určení složitosti v prostředí mimo TS:

Příklad 12.3 Uvažme následující implementaci **porovnávání řetězců**.

```
int str_cmp (int n, string a, string b) {
    int i;

    i = 0;
    while (i<n) {
        if (a[i] != b[i]) break;
        i++;
    }

    return (i==n);
}
```

- Pro určení složitosti aplikujme **uniformní cenové kritérium** např. tak, že budeme považovat složitost každého řádku programu (mimo deklarace) za jednotku. (Předpokládáme, že žádný cyklus není zapsán na jediném řádku.)
- **Složitost nejhoršího případu**

❖ Ilustrujme si nyní určení složitosti v prostředí mimo TS:

Příklad 12.4 Uvažme následující implementaci **porovnávání řetězců**.

```
int str_cmp (int n, string a, string b) {
    int i;

    i = 0;
    while (i<n) {
        if (a[i] != b[i]) break;
        i++;
    }

    return (i==n);
}
```

- Pro určení složitosti aplikujme **uniformní cenové kritérium** např. tak, že budeme považovat složitost každého řádku programu (mimo deklarace) za jednotku. (Předpokládáme, že žádný cyklus není zapsán na jediném řádku.)
- **Složitost nejhoršího případu lze pak snadno určit jako $3n + 3$:**
 - cyklus má 3 kroky, provede se n krát, tj. $3n$ kroků,
 - tělo funkce má 3 kroky (včetně testu ukončujícího cyklus).

Příklad 12.5 Uvažme následující implementaci řazení metodou insert-sort.

```
void insertsort(int n, int a[]) {
    int i, j, value;
    for (i=0; i<n; i++) {
        value = a[i];
        j = i - 1;
        while ((j >= 0) && (a[j] > value)) {
            a[j+1] = a[j];
            j = j - 1;
        }
        a[j+1] = value;
    }
}
```

- Pro určení složitosti aplikujme **uniformní cenové kritérium** např. tak, že budeme považovat složitost každého řádku programu (mimo deklarace) za jednotku. (Předpokládáme, že žádný cyklus není zapsán na jediném řádku.)
- **Složitost nejhoršího případu**

Příklad 12.6 Uvažme následující implementaci řazení metodou insert-sort.

```
void insertsort(int n, int a[]) {
    int i, j, value;
    for (i=0; i<n; i++) {
        value = a[i];
        j = i - 1;
        while ((j >= 0) && (a[j] > value)) {
            a[j+1] = a[j];
            j = j - 1;
        }
        a[j+1] = value;
    }
}
```

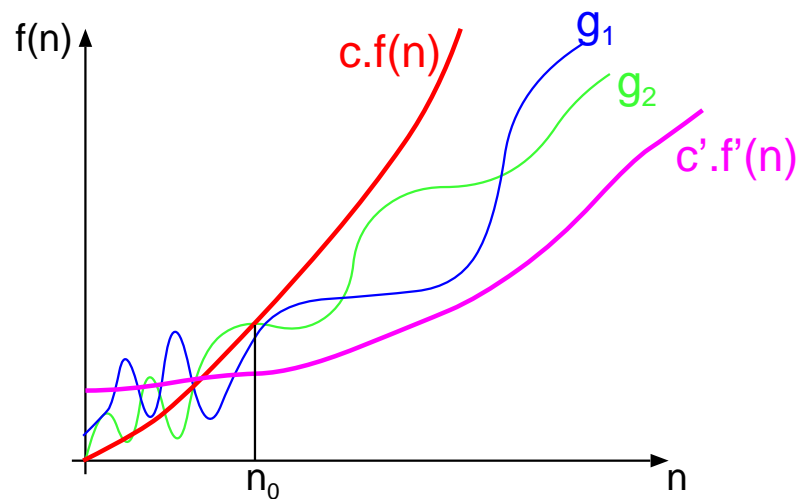
- Pro určení složitosti aplikujme **uniformní cenové kritérium** např. tak, že budeme považovat složitost každého řádku programu (mimo deklarace) za jednotku. (Předpokládáme, že žádný cyklus není zapsán na jediném řádku.)
- **Složitost lze pak určit jako $1.5n^2 + 3.5n + 1$:**
 - vnitřní cyklus má 3 kroky, provede se $0, 1, \dots, n - 1$ krát, tj.
 $3(0 + 1 + \dots + n - 1) = 3 \frac{n}{2}(0 + n - 1) = 1.5n^2 - 1.5n$ kroků,
 - vnější cyklus má mimo vnitřní cyklus 5 kroků (včetně testu ukončujícího vnitřní cyklus) a provede se n krát, tj. $5n$ kroků,
 - jeden krok připadá na ukončení vnějšího cyklu.

Asymptotická omezení složitosti

- ❖ Při popisu složitosti algoritmů (výpočtů TS), chceme často **vyločit vliv aditivních a multiplikativních konstant**:
 - různé aditivní a multiplikativní konstanty vzniknou velmi snadno „drobnými“ úpravami uvažovaných algoritmů,
 - např. srovnávání dvou řetězců je možné donekonečna zrychlovat tak, že budeme porovnávat současně 2, 3, 4, ... za sebou jdoucích znaků,
 - lineární komprese prostoru (rozšíření abecedy) a zrychlení výpočtu (před-vypočítání výsledků)
 - tyto úpravy ovšem nemají zásadní vliv na rychlost nárůstu časové složitosti (ve výše uvedeném případě nárůst zůstane kvadratický),
 - navíc při analýze složitosti mimo prostředí TS se dopouštíme jisté nepřesnosti již zavedením různých cenových kritérií.
- ❖ Proto se často složitost popisuje pomocí tzv. **asymptotických odhadů složitosti** – viz další stránka.

Definice 12.3 Necht' \mathcal{F} je množina funkcí $f : \mathbb{N} \rightarrow \mathbb{N}$. Pro danou funkci $f \in \mathcal{F}$ definujeme množiny funkcí $O(f(n))$, $\Omega(f(n))$ a $\Theta(f(n))$ takto:

- **Asymptotické horní omezení** funkce $f(n)$ je množina $O(f(n)) = \{g(n) \in \mathcal{F} \mid \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N} \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow 0 \leq g(n) \leq c \cdot f(n)\}$.
- **Asymptotické dolní omezení** funkce $f(n)$ je množina $\Omega(f(n)) = \{g(n) \in \mathcal{F} \mid \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N} \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow 0 \leq c \cdot f(n) \leq g(n)\}$.
- **Asymptotické oboustranné omezení** funkce $f(n)$ je množina $\Theta(f(n)) = \{g(n) \in \mathcal{F} \mid \exists c_1, c_2 \in \mathbb{R}^+ \exists n_0 \in \mathbb{N} \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow 0 \leq c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n)\}$.



Příklad 12.7 S využitím asymptotických odhadů složitosti můžeme říci, že složitost našeho srovnání řetězců patří do $O(n)$ a složitost insert-sort do $O(n^2)$.

Amortizovaná složitost – ukázka

❖ Příklad: dynamicky alokovaná tabulka T (odebrání záznamu zneplatní řádek)

- neznámá velikost – dynamická realokace mění kapacitu
- při naplnění tabulky alokujeme větší tabulku a složky do ní přesuneme
- při vyprázdnění na určitou mez alokujeme menší tabulku a složky do ní přesuneme
- $\alpha(T)$ je poměr platných položek (velikost) ku kapacitě tabulky
- pokud je $\alpha(T) = 1$ vložení prvku vede k alokaci 2-krát větší tabulky a přesunu
- pokud je $\alpha(T) \leq 0.5$ odebrání prvku vede k alokaci 2-krát menší tabulky a přesunu

❖ Složitost nejhoršího případu

- složitost jedné operace vložení i odebrání je rovna n (velikost pole)
- složitost n operací je v nejhorším případě $O(n^2)$

❖ Amortizovaná složitost

- n operací vložení:
 - c_i – cena i -té operace vložení
 - $c_i = i$ pokud $i - 1$ je mocninou 2, jinak $c_i = 1$
 - $\sum_{i=1}^n c_i \leq n + \sum_{j=0}^{\lfloor \log n \rfloor} 2^j < n + 2n < 3n$

Amortizovaná složitost – ukázka 2

- existuje posloupnost n operací (vlození, odebrání) se složitostí $\Theta(n^2)$
 - střídavě přidáváme odebíráme prvky kolem faktoru $\alpha(T) = 0.5$
 - každá třetí operace má cenu n

❖ Existuje implementace s lineární amortizovanou složitostí?

- vyhneme tomu, že kapacita může oscilovat mezi dvěma hodnotami při malém počtu operací (viz výše)
- pokud je $\alpha(T) \leq 0.25$ odebrání prvku vede k alokaci 2-krát menšího pole a přesunu
- určete amortizovanou složitost libovolných n operací (hlavní myšlenka)
 - $2^k + 1$ operací vložení, kapacita 2^{k+1} , cena $< 3(2^k + 1)$
 - realokaci vyvolá $2^{k-1} + 1$ operací odebrání, kapacita je 2^k , cena $2^{k-1} + 2^k + 1$
 - další realokaci vyvolá $2^{k-1} + 1$ operací vložení, kapacita 2^{k+1} , cena $2^{k-1} + 2^k + 1$
 - celkově operací: $2^k + 1 + 2 * 2^{k-1} + 2 = 2^{k+1} + 3$
 - celkově cena: $2 * 2^{k-1} + 5 * 2^k + 5 = 6 * 2^k + 5 = 3 * 2^{k+1} + 5$
 - ukázali jsem, že n operací má cenu $O(n)$

Třídy složitosti

Složitost problémů

- ❖ Přejdeme nyní od složitosti konkrétních algoritmů (Turingových strojů) ke **složitosti problémů**.
- ❖ Třídy složitosti zavádíme jako **prostředek ke kategorizaci (vytvoření hierarchie) problémů dle jejich složitosti**, tedy dle toho, jak dalece efektivní algoritmy můžeme navrhnout pro jejich rozhodování (u funkcí můžeme analogicky mluvit o složitosti jejich vyčíslování).
- ❖ Podobně jako u určování typu jazyka se budeme snažit **zařadit problém do co nejnižší třídy složitosti** – tedy určit složitost problému jako složitost jeho nejlepšího možného řešení.
 - Omezením této snahy je to, že (jak uvidíme) existují problémy, jejichž řešení je možné do nekonečna *významně* zrychlovat.
- ❖ Zařazení **různých problémů do téže třídy složitosti** může odhalit různé vnitřní podobnosti těchto problémů a může umožnit řešení jednoho převodem na druhý (a pragmatické využití např. různých již vyvinutých nástrojů).

Třídy složitosti

Definice 12.4 Mějme dány funkce $t, s : \mathbb{N} \rightarrow \mathbb{N}$ a necht' T_M , resp. S_M , značí časovou, resp. prostorovou, složitost TS M . Definujeme následující **časové a prostorové třídy složitosti deterministických a nedeterministických TS**:

- $DTime[t(n)] = \{L \mid \exists k\text{-páskový DTS } M : L = L(M) \text{ a } T_M \in O(t(n))\}$.
- $NTime[t(n)] = \{L \mid \exists k\text{-páskový NTS } M : L = L(M) \text{ a } T_M \in O(t(n))\}$.
- $DSpace[s(n)] = \{L \mid \exists k\text{-páskový DTS } M : L = L(M) \text{ a } S_M \in O(s(n))\}$.
- $NSpace[s(n)] = \{L \mid \exists k\text{-páskový NTS } M : L = L(M) \text{ a } S_M \in O(s(n))\}$.

❖ Definici tříd složitosti pak přímočaře **zobecňujeme tak, aby mohly být založeny na množině funkcí**, nejen na jedné konkrétní funkci.

❖ *Poznámka:* Dále ukážeme, že použití více pásek přináší jen polynomiální zrychlení. Na druhou stranu ukážeme, že zatímco nedeterminismus nepřináší nic podstatného z hlediska vyčíslitelnosti, může přinášet mnoho z hlediska složitosti.

Časově/prostorově zkonstruovatelné funkce

❖ Třídy složitosti obvykle budujeme nad tzv. **časově/prostorově zkonstruovatelnými funkcemi**:

- Důvodem zavedení časově/prostorově zkonstruovatelných funkcí je dosáhnout **intuitivní hierarchické struktury** tříd složitosti – např. odlišení tříd $f(n)$ a $2^{f(n)}$, což, jak uvidíme, pro třídy založené na obecných rekurzivních funkcích nelze.

Definice 12.5 Funkci $t : \mathbb{N} \rightarrow \mathbb{N}$ nazveme **časově zkonstruovatelnou** (time constructible), jestliže existuje vícepáskový TS, jenž pro libovolný vstup w zastaví po přesně $t(|w|)$ krocích.

Definice 12.6 Funkci $s : \mathbb{N} \rightarrow \mathbb{N}$ nazveme **prostorově zkonstruovatelnou** (space constructible), jestliže existuje vícepáskový TS, jenž pro libovolný vstup w zastaví s využitím přesně $s(|w|)$ buněk pásky.

❖ *Poznámka:* Pokud je jazyk L nad Σ přijímán strojem v čase/prostoru omezeném časově/prostorově zkonstruovatelnou funkcí, pak je také přijímán strojem, který pro každé $w \in \Sigma^*$ vždy zastaví:

- U časového omezení $t(n)$ si stačí předem spočítat, jaký je potřebný počet kroků a zastavit po jeho vyčerpání.
- U prostorového omezení spočteme z $s(n)$, $|Q|$, $|\Delta|$ maximální počet konfigurací, které můžeme vidět a z toho také plyne maximální počet kroků, které můžeme udělat, aniž bychom cyklili.

Nejběžněji užívané třídy složitosti

❖ Deterministický/nedeterministický polynomiální čas:

$$\mathbf{P} = \bigcup_{k=0}^{\infty} DTime(n^k)$$

$$\mathbf{NP} = \bigcup_{k=0}^{\infty} NTime(n^k)$$

❖ Deterministický/nedeterministický polynomiální prostor:

$$\mathbf{PSPACE} = \bigcup_{k=0}^{\infty} DSpace(n^k)$$

\equiv

$$\mathbf{NPSPACE} = \bigcup_{k=0}^{\infty} NSpace(n^k)$$

❖ *Poznámka:* Problémy ze třídy **PSPACE** se často reálně neřeší v polynomiálním prostoru – zvyšují se prostorové nároky výměnou za alespoň částečné snížení časových nároků (např. z $O(2^{n^2})$ na $O(2^n)$ u LTL model checkingu apod.). Intuitivně: je možno si pamatovat více mezivýsledků a není třeba je znovu spočítat.

Třídy pod a nad polynomiální složitostí

❖ Deterministický/nedeterministický logaritmický prostor:

$$\mathbf{LOGSPACE} = \bigcup_{k=0}^{\infty} DSpace(k \lg n)$$

$$\mathbf{NLOGSPACE} = \bigcup_{k=0}^{\infty} NSpace(k \lg n)$$

❖ Deterministický/nedeterministický exponenciální čas:

$$\mathbf{EXP} = \bigcup_{k=0}^{\infty} DTime(2^{n^k})$$

$$\mathbf{NEXP} = \bigcup_{k=0}^{\infty} NTime(2^{n^k})$$

❖ Deterministický/nedeterministický exponenciální prostor:

$$\mathbf{EXPSPACE} = \bigcup_{k=0}^{\infty} DSpace(2^{n^k})$$

≡

$$\mathbf{NEXPSPACE} = \bigcup_{k=0}^{\infty} NSpace(2^{n^k})$$

Třídy nad exponenciální složitostí

❖ Det./nedet. k -exponenciální čas/prostor založený na věži exponenciál $2^{2^{\dots^2}}$ o výšce k :

$$\mathbf{k\text{-EXP}} = \bigcup_{l=0}^{\infty} DTime(2^{2^{\dots^{2^{n^l}}}})$$

$$\mathbf{k\text{-NEXP}} = \bigcup_{l=0}^{\infty} NTime(2^{2^{\dots^{2^{n^l}}}})$$

$$\mathbf{k\text{-EXPSPACE}} = \bigcup_{l=0}^{\infty} DSpace(2^{2^{\dots^{2^{n^l}}}}) \equiv \mathbf{k\text{-NEXPSPACE}} = \bigcup_{l=0}^{\infty} NSpace(2^{2^{\dots^{2^{n^l}}}})$$

$$\mathbf{ELEMENTARY} = \bigcup_{k=0}^{\infty} \mathbf{k\text{-EXP}}$$

Vrchol hierarchie tříd složitosti

❖ Na vrcholu hierarchie tříd složitosti se pak hovoří o obecných třídách jazyků (funkcí), se kterými jsme se již setkali:

- třída primitivně-rekurzivních funkcí **PR** (implementovatelných pomocí zanořených cyklů s pevným počtem opakování – `for i=... to ...`),
- třída μ -rekurzivních funkcí (rekurzivních jazyků) **R** (implementovatelných pomocí cyklů s předem neurčeným počtem opakování – `while ...`) a
- třída rekurzivně vyčíslitelných funkcí (rekurzivně vyčíslitelných jazyků) **RE**.

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.**

Některé další zajímavé výsledky

Věta 12.9 (Blumův teorém) Pro každou totální vyčíslitelnou funkci $f : \mathbb{N} \rightarrow \mathbb{N}$ existuje problém, jehož každé řešení s nějakou složitostí $t(n)$ může být zlepšeno tak, že nové řešení má složitost $f(t(n))$ pro skoro každé $n \in \mathbb{N}$.

Důkaz. Viz literatura. □

❖ V důsledku Blumova teorému tedy **existují problémy** jejichž řešení se složitostí $t(n)$ je možné donekonečna zrychlovat na $\lg t(n)$, $\lg \lg t(n)$, $\lg \lg \lg t(n)$, ...

❖ Nutnost pracovat s **časově zkonstruovatelnými funkcemi**, abychom se vyhnuli např. tomu, že $DTime[f(n)] = DTime[2^{f(n)}]$ pro nějaké $f(n)$, vyjadřuje následující teorém:

Věta 12.10 (Gap Theorem) Ke každé rekurzivní funkci $\phi(n) > n$ existuje rekurzivní funkce $f(n)$ taková, že $DTime[\phi(f(n))] = DTime[f(n)]$.

Důkaz. Viz literatura – založeno na konstrukci funkce f takové, že žádný TS nezastaví pro vstup délky n počtem kroků mezi $f(n)$ a $\phi(f(n))$. □

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 12.7 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 12.8 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 12.9 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.

❖ Pro třídy $\mathcal{C}_1 \subseteq \mathcal{C}_2$ a jazyk L , jenž je \mathcal{C}_2 -úplný vůči \mathcal{R} redukovatelnosti, platí, že buď \mathcal{C}_2 je celá \mathcal{R} redukovatelná na \mathcal{C}_1 , nebo $L \in \mathcal{C}_2 \setminus \mathcal{C}_1$.

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 12.10 *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 12.11 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 . \square

Příklad 12.8 Funkce $f : \{x, y\}^* \rightarrow \{x, y, z\}^*$ definována jako $f(v) = vzv$ je polynomiální redukcí jazyka $L_1 = \{w \mid w \text{ je palindrom nad } \{x, y\}\}$ na jazyk $L_2 = \{wzw^R \mid w \in \{x, y\}^*\}$.

❖ Předchozí věta nám dává praktickou možnost jak ukázat, že určitý jazyk je ve třídě P. Navíc, přeformulujeme-li tuto větu takto:

„Jestliže platí $L_1 \leq_P^m L_2$ a L_1 neleží v P , pak L_2 také neleží v P “,
můžeme dokazovat, že určitý jazyk neleží 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, částečně rozhodnutelná nespłnitelnost,
- **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:

- 3-SAT a obecný SAT – viz dále,
- řada grafových problémů, např.:
 - existence kliky dané velikosti,
 - existence Hamiltonovské kružnice v neorientovaném grafu,
 - existence orientované Hamiltonovské kružnice v orientovaném grafu,
 - barvitelnost – lze daný neorientovaný graf obarvit určitým počtem barev?,
 - uzlové pokrytí neorientovaného grafu množinou uzlů o určité velikosti (tj. množinou uzlů, se kterou souvisí všechny hrany),
 - ...
- 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 kvantové počítače.

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.

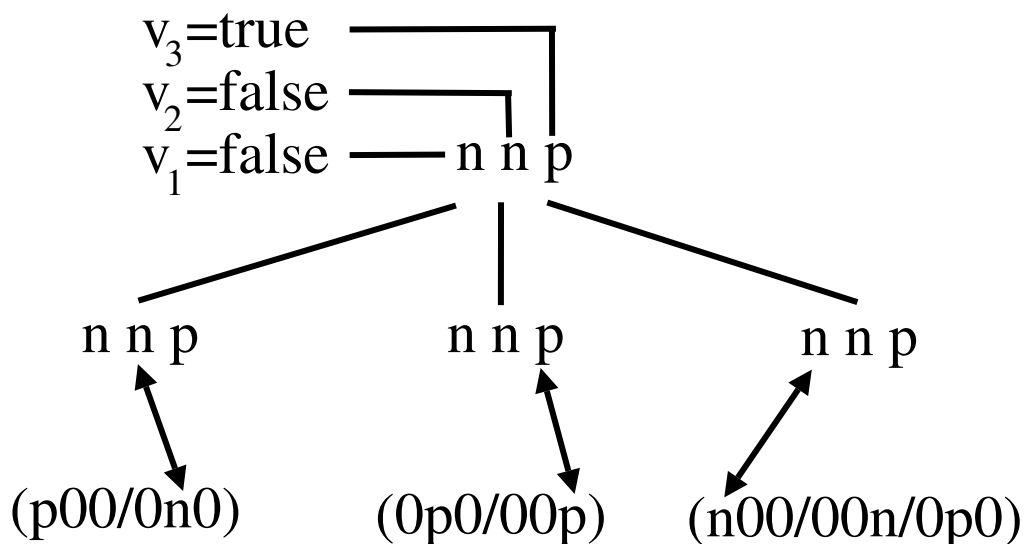
Příklad 12.9 SAT-problém obsahuje proměnné v_1, v_2, v_3 a klausule $v_1 \vee \overline{v_2}$, $v_2 \vee v_3$, $\overline{v_1} \vee \overline{v_3} \vee v_2$ bude reprezentována řetězcem: $(p00/0n0)(0p0/00p)(n00/00n/0p0)$.

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

Důkaz. 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. Jádro 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?