68. Časová a paměťová složitost (asymptotická a amortizovaná složitost, třídy složitosti, úplnost, SAT problém).

Časová a prostorová složitost Turingových strojů

Ať $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ je *k*-páskový **deterministický** Turingův stroj. Řekneme, že stroj M **přijímá jazyk** L **v čase** $T_M: \mathbb{N} \to \mathbb{N}$, pokud $L = L(M)$ a stroj M přijme každý řetězec $w \in L$ nejvýše v $T_M(|w|)$ krocích. Zobrazení $T_M: \mathbb{N} \to \mathbb{N}$ je **časová složitost** stroje M.

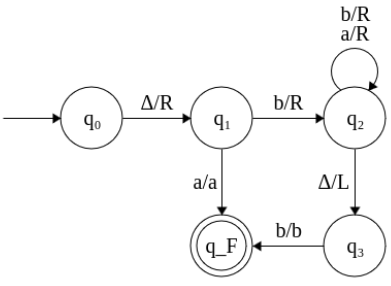
Ať $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ je *k*-páskový **nedeterministický** Turingův stroj. Řekneme, že stroj M **přijímá jazyk** L **v čase** $T_M: \mathbb{N} \to \mathbb{N}$, pokud $L = L(M)$ a stroj M **může přijmout** každý řetězec $w \in L$ nejvýše v $T_M(|w|)$ krocích. Zobrazení $T_M: \mathbb{N} \to \mathbb{N}$ je **časová složitost** stroje M.

Ať $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ je *k*-páskový **deterministický** Turingův stroj. Řekneme, že stroj M **přijímá jazyk** L **v prostoru** $S_M: \mathbb{N} \to \mathbb{N}$, pokud $L = L(M)$ a stroj M přijme každý řetězec $w \in L$ za použití nejvýše $S_M(|w|)$ buněk pásky. Zobrazení $S_M: \mathbb{N} \to \mathbb{N}$ je **prostorová složitost** stroje M.

Ať $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ je *k*-páskový **nedeterministický** Turingův stroj. Řekneme, že stroj M **přijímá jazyk** L **v prostoru** $S_M: \mathbb{N} \to \mathbb{N}$, pokud $L = L(M)$ a stroj M **může přijmout** každý řetězec $w \in L$ za použití nejvýše $S_M(|w|)$ buněk pásky. Zobrazení $S_M: \mathbb{N} \to \mathbb{N}$ je **prostorová složitost** stroje M.

Pozn.: V kontextu prostorové složitosti považujeme za použité pouze ty buňky pásky, na nichž spočinula hlava stroje (tzn. nikoliv ty, na nichž byl zapsán vstup, který nebyl nikdy přečten).

Příklad.
Mějme deterministický Turingův stroj $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ zobrazený níže:



Tento stroj přijímá jazyk $L(M) = \{aw \mid w \in \{a, b\}^*\} \cup \{wb \mid w \in \{a, b\}^*\}$, tedy jazyk takových řetězců, které začínají symbolem a nebo končí symbolem b.

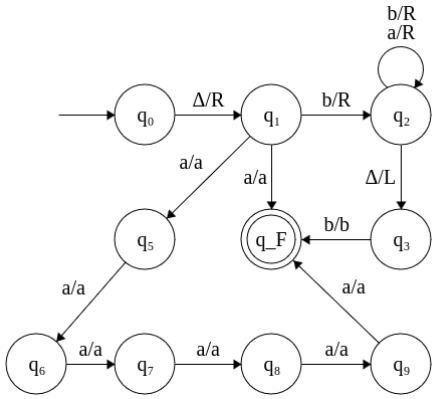
Zřejmě platí:

- řetězec aaa je přijat ve dvou krocích
  - $(q_0, \Delta aaa\Delta\Delta^\omega, 0) \vdash (q_1, \Delta aaa\Delta\Delta^\omega, 1) \vdash (q_F, \Delta aaa\Delta\Delta^\omega, 1)$
  - stroj vykoná dva kroky a akceptuje
- řetězec bbb je přijat v šesti krocích
  - $(q_0, \Delta bbb\Delta\Delta^\omega, 0) \vdash (q_1, \Delta bbb\Delta\Delta^\omega, 1) \vdash (q_2, \Delta bbb\Delta\Delta^\omega, 2) \vdash (q_2, \Delta bbb\Delta\Delta^\omega, 3)$
    $\vdash (q_2, \Delta bbb\Delta\Delta^\omega, 4) \vdash (q_3, \Delta bbb\Delta\Delta^\omega, 3) \vdash (q_F, \Delta bbb\Delta\Delta^\omega, 3)$
  - stroj vykoná šest kroků a akceptuje
- řetězec aaa je přijat za použití 2 buněk pásky
  - využívá pouze úvodní symbol Δ a 1 buňku pásky pro vstup
- řetězec bbb je přijat za použití 5 buněk pásky
  - stroj používá 3 buňky pásky pro vstup, jednu buňku pro výchozíΔ a hlava stroje spočine na jednom dalším symbolu Δ za vstupem
- na všech osmi vstupech délky 3 se stroj chová následovně:

vstup	počet kroků	počet použitých buněk
aaa	2	2
aab	2	2
aba	2	2
abb	2	2
baa	5	5
bab	6	5
bba	5	5
bbb	6	5

  - označíme (zde žlutě) ta slova, která patří do jazyka $L(M)$
  - nejvíce kroků stroj vykoná na akceptovaných vstupech bab, bbb, v obou případech jde o 6 kroků
  - z toho důvodu $T_M(3) = 6$, neboť jde o maximum napříč vstupy o délce 3
  - analogicky $S_M(3) = 5$, jde o maximální hodnotu ze všech počtů použitých buněk napříč vstupy délky 3
- obecně
  - $T_M(n) = n + 3$
  - $S_M(n) = n + 2$

Předpokládejme dále, že by výše zakreslený Turingův stroj byl nedeterministický, jak naznačuje toto schéma:



Ze stavu q_1 zde vede alternativní cesta pro přijetí řetězců začínajících symbolem a. Zkoumejme, jak se stroj chová na vstupech délky 3:

vstup	počet kroků	počet použitých buněk
aaa	$2, 7$	2
aab	$2, 7$	2
aba	$2, 7$	2
abb	$2, 7$	2
baa	5	5
bab	6	5
bba	5	5
bbb	6	5

Všimněme si, že:

- pro akceptování některých slov existuje více alternativních cest
- řádky popisující slova začínající na a obsahují více možností pro počet kroků potřebný k přijetí daného slova
- při vyhodnocování $T_M(3)$
  - na každém řádku odpovídajícímu akceptovanému slovu ve sloupci *počet kroků* najdeme **minimální** hodnotu, tedy hodnotu popisující nejlepší cestu k přijetí daného slova
  - z těchto minimálních hodnot najdeme **maximální** hodnotu napříč všemi slovy
  - tato hodnota odpovídá $T_M(3)$
- $T_M(3) = 6$, ačkoliv pro některá slova existovaly cesty vyžadující 7 kroků k akceptování

Asymptotická omezení

Ať $\mathcal{F}$ je třída všech funkcí typu $\mathbb{N} \to \mathbb{N}$. Dále ať $f \in \mathcal{F}$ je zobrazení typu $\mathbb{N} \to \mathbb{N}$.
Asymptotické horní omezení pro funkci f je třída funkcí

$$O(f(n)) = \{g(n) \in \mathcal{F} \mid \exists c \in \mathbb{R}^+ : \exists n_0 \in \mathbb{N} : \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow 0 \leq g(n) \leq c \cdot f(n)\}$$

Jde tedy o třídu funkcí, jež je možné shora ohraničit funkcí $f(n)$ při zanedbání aditivních a multiplikativních konstant. Pokud je pro všechna přirozená čísla n o minimální hodnotě n_0 možné funkci $g(n)$ shora ohraničit funkcí $f(n)$ pronásobenou libovolnou kladnou konstantou, pak $f(n)$ asymptoticky shora ohraničuje funkci $g(n)$.

Příklad.
Mějme zobrazení $f(n) = n^2 - 10$. Následující funkce patří do $O(f(n))$:

- $g(n) = n$
  - jelikož $\lim_{n \to \infty} \frac{c \cdot (n^2 - 10)}{n} = \infty$ pro libovolné kladné reálné c, pak určitě existuje vhodné n_0
  - lineární funkci je možné asymptoticky shora ohraničit kvadratickou funkcí
- $g(n) = n^2$
  - jelikož $\lim_{n \to \infty} \frac{c \cdot (n^2 - 10)}{n^2} > 1$ např. pro $c = 2$, pak určitě existuje vhodné n_0
  - kvadratickou funkci s kladným kvadratickým členem je možné asymptoticky shora ohraničit libovolnou jinou kvadratickou funkcí s kladným kvadratickým členem
  - aditivní konstanta -10 zde nehraje roli
- $g(n) = 10000n^2$
  - jelikož $\lim_{n \to \infty} \frac{c \cdot (n^2 - 10)}{10000n^2} > 1$ např. pro $c = 20000$, pak určitě existuje vhodné n_0
  - multiplikativní konstanta 10000 zde nehraje roli

Následující funkce nepatří do $O(f(n))$:

- $g(n) = n^3$
  - jelikož $\lim_{n \to \infty} \frac{c \cdot (n^2 - 10)}{n^3} = 0$ nehledě na zvolené kladné reálné c, pak určitě neexistuje vhodné n_0
  - kubická funkce roste asymptoticky rychleji než funkce kvadratická, přičemž žádná aditivní ani multiplikativní konstanta tento rozdíl asymptoticky nepokryje

Asymptotické dolní omezení pro funkci f je třída funkcí

$$\Omega(f(n)) = \{g(n) \in \mathcal{F} \mid \exists c \in \mathbb{R}^+ : \exists n_0 \in \mathbb{N} : \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow 0 \leq c \cdot f(n) \leq g(n)\}$$

Amortizovaná složitost

Amortizovaná složitost slouží k analýze **průměrné složitosti sekvence n operací v nejhorším případě**. Bere přitom v potaz sekvence operací v takové podobě, v jaké mohou být reálně prováděny. Pokud má některá operace v nejhorším případě velkou časovou složitost, která se ovšem reálně projeví jen vzácně, v případě analýzy amortizované složitosti s něčím takovým počítáme.

Pro analýzu amortizované složitosti využíváme následující techniky: metoda seskupování a metoda účtů.

Metoda seskupování

- jednotlivé operace rozdělujeme do skupin a analyzujeme složitost celých skupin zvlášť

Metoda účtů

- předpokládáme existenci účtu, jehož zůstatek se mění v závislosti na prováděných operacích
- každá operace je ohodnocena cenou a kredity
- cena *c* nějaké operace určuje hodnotu, kterou je třeba zaplatit za provedení operace
- kredity *k* nějaké operace určují prostředky, které je možné za provedení operace uhradit, aniž bychom je museli odebrat z účtu
- pokud $c \geq k$, pak je třeba zaplatit $c - k$ jednotek z účtu
- pokud $c \leq k$, pak je zůstatek na účtu zvýšen o $k - c$ jednotek
- pokud je zůstatek na účtu z, pak obecně dojde k jeho změně na hodnotu $z + k - c$ po provedení operace s cenou c a s kredity k
- vložení přebývajících kreditů na účet představuje předplácení možnosti provedení drahé operace v budoucnu
- odebrání přebytečných kreditů z účtu odpovídá provedení drahé operace, která byla jinými operacemi předplacena v minulosti
- **na začátku provádění posloupnosti operací je zůstatek na účtu nulový**
- pokud je během celé posloupnosti operací zůstatek nezáporný, pak součet kreditů všech vykonaných operací je vyšší než celková složitost dané sekvence operací

Příklad.
Mějme datovou strukturu pro frontu, která umožňuje provedení následujících operací:

- **ENQ**
  - vložení nového elementu na konec fronty
- **DEQ**

    - odstranění elementu z čela fronty a jeho současné vytisknutí
    - nemá žádný efekt při provedení nad prázdnou frontou
- **CLEAR**
    - odstranění a současné vytisknutí všech elementů fronty (po jednom, jako opakované provádění DEQ)
    - vede k vyprázdnění celé fronty

Počet elementů fronty Q zapisujeme jako $|Q|$ nebo *n*. Začínáme s prázdnou frontou. Ceny a kredity jednotlivých operací definujeme následovně:

operace	cena	kredity
ENQ	1	2
DEQ	$min\{1, \|Q\|\}$	0
CLEAR	$\|Q\|$	0

- **ENQ**
    - cena 1 odpovídá vložení 1 elementu do fronty
    - hodnotíme 2 kredity, neboť
        - 1 kredit zaplatí aktuálně prováděné vkládání elementu
        - 1 kreditem předplácíme možnost pozdějšího odebrání právě vloženého elementu z fronty
- **DEQ**
    - cena 1 odpovídá odebrání 1 elementu z fronty
    - pokud je ovšem fronta prázdná, je cena nulová
    - hodnotíme 0 kredity, neboť
        - 1 kredit potřebný pro uhrazení ceny musel být v minulosti vložen na účet, když byl předmětný prvek vkládán do fronty
        - při odebrání elementu je tento kredit uhrazen
- **CLEAR**
    - cena $|Q|$ odpovídá odebrání $|Q|$ elementů z fronty
    - hodnotíme 0 kredity, neboť
        - $|Q|$ kreditů potřebných pro uhrazení ceny muselo být v minulosti vloženo na účet, neboť do fronty bylo umístěno daných $|Q|$ elementů, přičemž došlo k předplacení pozdějšího provedení operace **CLEAR**
        - při odstranění elementů z fronty jsou tyto kredity odebrány z účtu

Naivní analýza složitosti posloupnosti *n* operací by vedla k výsledku $O(n^2)$, neboť operace smazání fronty má lineární složitost vzhledem k velikosti fronty a mohla by být provedena *n*-krát. Analýza amortizované složitosti ovšem ukáže, že složitost je ve skutečnosti jiná. Provedení *n* operací **CLEAR** má reálně nižší složitost, protože

po prvním vykonání je fronta vyprázdněná a následně je operace volána nad prázdnou frontou.

Operace **ENQ** je ohodnocena nejvyšším množstvím kreditů (2). Maximální počet kreditů na účtu po provení *n* operací je tedy *2n.* Zůstatek na účtu je současně v každém okamžiku nezáporný. Z toho důvodu je složitost provedení posloupnosti *n* operací nejvýše *2n*.

Třídy složitosti

Pro zobrazení $f: \mathbb{N} \to \mathbb{N}$ definujeme následující třídy jazyků:

- $DTIME[f(n)]$
  - $DTIME[f(n)] = \{L \mid \exists k - páskový\ DTS\ M : L(M) = L \wedge T_M \in O(f(n))\}$
  - třída všech jazyků, které je možné přijímat k-páskovým **deterministickým** Turingovým strojem **v čase** $O(f(n))$
- $NTIME[f(n)]$
  - $NTIME[f(n)] = \{L \mid \exists k - páskový\ NTS\ M : L(M) = L \wedge T_M \in O(f(n))\}$
  - třída všech jazyků, které je možné přijímat k-páskovým **nedeterministickým** Turingovým strojem **v čase** $O(f(n))$
- $DSPACE[f(n)]$
  - $DSPACE[f(n)] = \{L \mid \exists k - páskový\ DTS\ M : L(M) = L \wedge S_M \in O(f(n))\}$
  - třída všech jazyků, které je možné přijímat k-páskovým **deterministickým** Turingovým strojem **v prostoru** $O(f(n))$
- $NSPACE[f(n)]$
  - $NSPACE[f(n)] = \{L \mid \exists k - páskový\ NTS\ M : L(M) = L \wedge S_M \in O(f(n))\}$
  - třída všech jazyků, které je možné přijímat k-páskovým **nedeterministickým** Turingovým strojem **v prostoru** $O(f(n))$

Příklad.

- $L = \{aw \mid w \in \{a,b\}^*\} \in DTIME[1]$
  - odpovídající Turingův stroj pouze ověří, zda je první symbol vstupu a
  - složitost nijak nezávisí na délce vstupu
- $L = \{aw \mid w \in \{a,b\}^*\} \in DTIME[n]$
  - jelikož daný jazyk je možné přijímat deterministicky v konstantním čase, pak jistě lze přijímat deterministicky v lineárním čase
  - $DTIME[1] \subseteq DTIME[n]$
- $L = \{wb \mid w \in \{a,b\}^*\} \in DTIME[n]$
  - odpovídající Turingův stroj musí projít celý vstup a ověřit, zda je poslední symbol roven b
- $L = \{wb \mid w \in \{a,b\}^*\} \notin DTIME[1]$
  - nelze řešit v konstantním čase, neboť potřebujeme projít celý vsutp
- $L = \{w \in \{a,b\}^* \mid w > 10^{100!}!\} \in DTIME[1]$

    - pro žádný vstup není třeba vykonat více než přibližně $10^{100!}$! kroků, což je konstantní číslo (ale jestli naimplementujete podobný algoritmus s argumentací, že má konstantní složitost, kolegové vám do kafe nasypou sůl)

Dále pracujeme s následujícími třídami složitosti:

- **LOGSPACE**
    - $LOGSPACE = \cup_{k=0}^{\infty} \quad DSPACE[k \cdot ln(n)]$
    - třída jazyků, které je možné přijímat deterministickým Turingovým strojem, který pracuje v logaritmickém prostoru
- **NLOGSPACE**
    - $NLOGSPACE = \cup_{k=0}^{\infty} \quad NSPACE[k \cdot ln(n)]$
    - třída jazyků, které je možné přijímat nedeterministickým Turingovým strojem, který pracuje v logaritmickém prostoru
- **P**
    - $P = \cup_{k=0}^{\infty} \quad DTIME[n^k]$
    - třída jazyků, které je možné přijímat deterministickým Turingovým strojem, který pracuje v polynomiálním čase
- **NP**
    - $NP = \cup_{k=0}^{\infty} \quad NTIME[n^k]$
    - třída jazyků, které je možné přijímat nedeterministickým Turingovým strojem, který pracuje v polynomiálním čase
- **PSPACE**
    - $PSPACE = \cup_{k=0}^{\infty} \quad DSPACE[n^k]$
    - třída jazyků, které je možné přijímat deterministickým Turingovým strojem, který pracuje v polynomiálním prostoru
- **NPSPACE**
    - $NPSPACE = \cup_{k=0}^{\infty} \quad NSPACE[n^k]$
    - třída jazyků, které je možné přijímat nedeterministickým Turingovým strojem, který pracuje v polynomiálním prostoru
- **EXP**
    - $EXP = \cup_{k=0}^{\infty} \quad DTIME\left[2^{n^k}\right]$
    - třída jazyků, které je možné přijímat deterministickým Turingovým strojem, který pracuje v exponenciálním čase
- **NEXP**
    - $NEXP = \cup_{k=0}^{\infty} \quad NTIME\left[2^{n^k}\right]$
    - třída jazyků, které je možné přijímat nedeterministickým Turingovým strojem, který pracuje v exponenciálním čase
- **EXPSPACE**
    - $EXPSPACE = \cup_{k=0}^{\infty} \quad DSPACE\left[2^{n^k}\right]$
    - třída jazyků, které je možné přijímat deterministickým Turingovým strojem, který pracuje v exponenciálním prostoru
- **NEXPSPACE**

    - $NEXPSPACE = \cup_{k=0}^{\infty} \quad NSPACE\left[2^{n^k}\right]$
    - třída jazyků, které je možné přijímat nedeterministickým Turingovým strojem, který pracuje v exponenciálním prostoru
- **k-EXP**
    - $k - EXP = \cup_{l=0}^{\infty} \quad DTIME\left[{}^{k}2^{n^l}\right]$
    - *pozn.: zápis ${}^{k}2^{n^l}$ značí k-krát opakovanou exponenciaci základu 2, na jejímž vrcholu je další exponent n^l*
- **ELEMENTARY**
    - $ELEMENTARY = \cup_{k=0}^{\infty} \quad k - EXP$

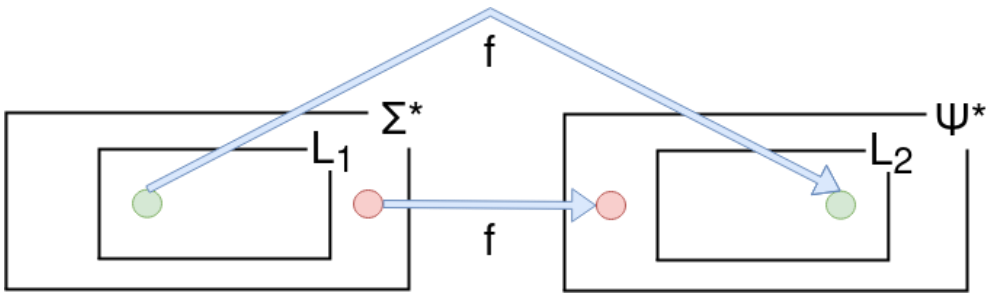
Mezi těmito třídami existují například následující vztahy:

- **LOGSPACE ⊆ NLOGSPACE ⊆ P ⊆ NP**
- **NP ⊆ PSPACE = NPSPACE ⊆ EXP ⊆ NEXP**
- **NEXP ⊆ EXPSPACE = NEXPSPACE ⊆ 2-EXP ⊆ 2-NEXP ⊆ …**
- **NLOGSPACE ⊂ PSPACE**
- **P ⊂ EXP**
- **NP ⊂ NEXP**
- **PSPACE ⊂ NEXPSPACE**
- **EXP ⊂ 2-EXP**

Úplnost jazyků vzhledem ke složitostním třídám

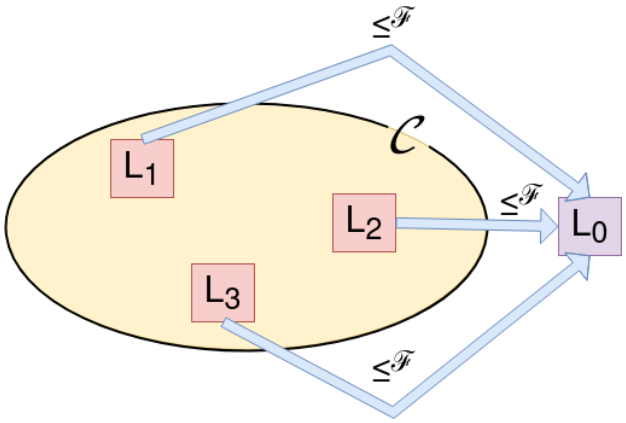
Ať $\mathcal{F}$ je třída funkcí a ať Σ, Ψ jsou abecedy. Dále ať L_1, L_2 jsou po řadě jazyky nad abecedami Σ, Ψ. Jazyk L_1 je **$\mathcal{F}$-redukovatelný** na jazyk L_2, pokud existuje takové zobrazení $f \in \mathcal{F}$, že $\forall w \in \Sigma^* : w \in L_1 \Leftrightarrow f(w) \in L_2$. Tuto skutečnost zapisujeme jako $L_1 \leq_{\mathcal{F}}^{m} L_2$.

Funkce ze třídy $\mathcal{F}$ je zde zobrazením z množiny Σ^* do množiny Ψ^*, přičemž je realizovatelná pomocí úplného Turingova stroje. Konkrétní typ redukovatelnosti je tedy typicky závislý na tom, jaká je časová nebo prostorová složitost odpovídajících Turingových strojů realizujících funkce ze třídy $\mathcal{F}$.

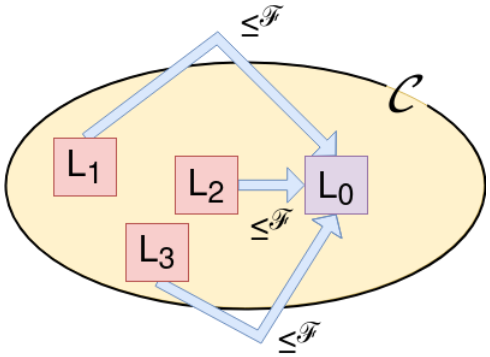


Ať $\mathcal{F}$ je třída funkcí a ať $\mathcal{C}$ je třída jazyků. Jazyk L_0 je **$\mathcal{C}$-těžký** vůči $\mathcal{F}$-redukovatelnosti, pokud $\forall L \in \mathcal{C} : L \leq_{\mathcal{F}}^{m} L_0$.

Jazyk L_0 je tedy $\mathcal{C}$-těžký vůči $\mathcal{F}$-redukovatelnosti, pokud je možné úplně každý jazyk ze třídy $\mathcal{C}$ na jazyk L_0 $\mathcal{F}$-redukovat.



Ať $\mathcal{F}$ je třída funkcí a $\mathcal{C}$ třída jazyků. Jazyk L_0 je **$\mathcal{C}$-úplný** vůči $\mathcal{F}$-redukovatelnosti, pokud $L_0 \in \mathcal{C}$ a L_0 je $\mathcal{C}$-těžký vůči $\mathcal{F}$-redukovatelnosti. Jazyk L_0 je tedy $\mathcal{C}$-úplný vůči $\mathcal{F}$-redukovatelnosti, pokud je možné úplně každý jazyk ze třídy $\mathcal{C}$ na jazyk L_0 $\mathcal{F}$-redukovat a současně L_0 patří do třídy $\mathcal{C}$.



Ať Σ, Ψ jsou abecedy. Dále ať L_1, L_2 jsou po řadě jazyky nad abecedami Σ, Ψ. Jazyk L_1 je **polynomiálně redukovatelný** na jazyk L_2, pokud existuje takové zobrazení f, že $\forall w \in \Sigma^* : w \in L_1 \Leftrightarrow f(w) \in L_2$ a současně je zobrazení f realizováno pomocí úplného Turingova stroje, jenž pracuje v polynomiálním čase. Zobrazení f je **polynomiální redukcí**.

Ať Σ je abeceda a ať L_0 je jazyk nad abecedou Σ. Jazyk L_0 je **NP-těžký** (vůči polynomiální redukovatelnosti), pokud je každý jazyk ze třídy NP polynomiálně redukovatelný na jazyk L_0.

Ať Σ je abeceda a ať L_0 je jazyk nad abecedou Σ. Jazyk L_0 je **NP-úplný** (vůči polynomiální redukovatelnosti), pokud je NP-těžký a současně patří do třídy NP.

SAT problém

Mějme kódování výrokových formulí nad abecedou {0, 1}. Definujeme jazyk SAT (jazyk splnitelných výrokových formulí následovně):

SAT = {<φ> | <φ> je kód výrokové formule φ, která je splnitelná}.

Pozn.: Dle některých definic uvažujeme pouze o výrokových formulích v konjunktivní normální formě. Pro každou výrokovou formuli lze však nalézt ekvisplnitelnou výrokovou formuli v konjunktivní normální formě v polynomiálním čase pomocí takzvané Tseytinovy transformace.

Výrokovou formuli považujeme za splnitelnou tehdy, pokud je možné přiřadit jejím výrokovým proměnným takové hodnoty výrokových konstant, aby byla daná formule vyhodnocena jako *true* (tedy formule je splnitelná právě tehdy, pokud má (existuje) model).

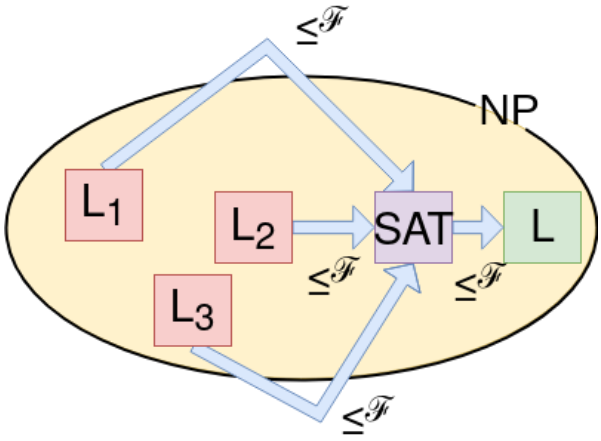
Příklad.
Mějme následující výrokové formule:

- $x_1 \vee x_2$
  - formule je splnitelná například v případě, kdy proměnným x_1, x_2 přiřadíme hodnotu *true*
- $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$
  - formule je nesplnitelná

Pro SAT problém platí následující:

- patří do třídy **NP**
  - existuje nedeterministický Turingův stroj pracující v polynomiálním čase, který akceptuje SAT
  - takový stroj nejprve nedeterministicky vybere přiřazení booleovských konstant jednotlivým výrokovým proměnným (odhadne model)
  - následně v polynomiálním čase zkontroluje, zda je s tímto přiřazením formule vyhodnocena jako *true*
  - pokud ano, akceptuje, jinak odmítne
  - existuje-li vhodná nedeterministická volba přiřazení booleovských konstrant, pak existuje výpočetní větev stroje, která je akceptující, jinak neexistuje
- je **NP-úplný** vzhledem k polynomiální redukovatelnosti
- jde o první problém, u nějž bylo dokázáno, že je **NP-úplný**
  - důkaz NP-těžkosti provedl Stephen Cook tak, že obecně pro každý jazyk ze třídy NP ukázal jeho polynomiální redukovatelnost na SAT
- skrze problém SAT lze snadno dokázat NP-těžkost dalších jazyků
  - jelikož je problém SAT NP-těžký vzhledem k polynomiální redukovatelnosti, pak je možné na SAT polynomiálně redukovat každý jazyk ze třídy NP

- pokud tedy polynomiálně redukujeme SAT na jiný jazyk L, pak je L jistě také NP-těžký
- to plyne z tranzitivity relace polynomiální redukovatelnosti



Pokud v textu najdete chybu, nebudete něčemu rozumět nebo budete mít dojem, že by bylo vhodné něco doplnit, kontaktujte na discordu uživatele kocotom.