Výpočetní složitost I

pro obor logika na FF UK

Petr Savický

1 Úvod

Složitostí algoritmické úlohy se rozumí především její časová a paměťová náročnost při řešení na počítači. Časová náročnost se měří počtem kroků, které algoritmus musí k řešení úlohy provést. Paměťová náročnost se obvykle nazývá prostorovou složitostí a měří se počtem znaků z nějaké konečné abecedy, které algoritmus potřebuje mít uloženy v paměti počítače. Protože časové i paměťové nároky obvykle rostou s rostoucí délkou popisu řešené instance úlohy, vyjadřuje se časová i prostorová složitost jako funkce délky vstupu.

Při studiu složitosti je možné zkoumat složitost konkrétního algoritmu pro danou úlohu nebo zkoumat složitost úlohy samotné, což lze chápat například tak, že chceme odhad složitosti algoritmu, který je v nějakém smyslu pro danou úlohu nejlepší možný. Pro řadu konkrétních algoritmů je možné jejich složitost zjistit nebo alespoň dobře odhadnout. Zjišťování složitosti úloh, tj. nejmenší nutné složitosti algoritmu pro její řešení, je problém podstatně obtížnější, protože vyžaduje kvantifikaci přes všechny možné algoritmy pro danou úlohu. Již definice pojmu složitost úlohy vyžaduje určité zjednodušení. Protože je složitost algoritmů vyjadřována jako funkce délky vstupu a funkce nejsou lineárně uspořádané, není minimální složitost algoritmu pro danou úlohu dobře definovaný pojem. Pro účely teoretického zkoumání složitosti se proto složitosti algoritmů porovnávají asymptoticky, tedy pro vstupy, jejichž délka neomezeně roste. Existuje řada úloh, jejichž složitost lze i při tomto zjednodušení v současné době charakterizovat jen nepřímo, například odkazem na celou třídu úloh podobné složitosti, aniž bychom jejich skutečnou složitost znali. K tomuto účelu byla vytvořena hierarchie tříd úloh, které se nazývají třídami složitosti. Typickými příklady těchto tříd jsou $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE}$.

Třída P je třídou úloh, které lze řešit v čase, který je omezen polynomem od délky vstupu. Tato třída je považována za formalizaci pojmu efektivně řešitelná úloha. Klasifikace algoritmů na polynomiální a (alespoň) exponenciální je v jednoduchých případech přirozená. Například test splnitelnosti Hornovské formule nebo úloha nalezení nejkratší cesty v grafu mají polynomiální složitost, zatímco pro test splnitelnosti obecné výrokové formule v konjunktivní normální formě nebo nalezení maximální kliky[1] v grafu jsou známy pouze exponenciální algoritmy. Třída P se využívá jako obecná definice efektivní vyčíslitelnosti, protože tyto výchozí známé příklady zobecňuje konzistentním způsobem. Definice

[1] Klika v grafu je množina vrcholů, z nichž každé dva jsou spojeny hranou.

třídy P sice formálně závisí na výpočetním modelu, ale pro všechny běžně používané modely, speciálně pro běžné varianty Turingova stroje a RAM (random access machine), vede na tutéž třídu úloh. V důkazu těchto ekvivalencí hraje podstatnou roli fakt, že algoritmy s polynomiální složitostí jsou uzavřeny na skládání algoritmů ve smyslu volání podprogramů.

Pro praktické účely je pochopitelně potřeba uvažovat složitost algoritmů přesněji než pouze rozlišením, zda mají polynomiální složitost nebo ne. Například algoritmus složitosti n^{100}, kde n je délka vstupu, nelze použít pro vstupy již relativně malé délky. Uvažme dále například polynomiální algoritmus složitosti $1.3 \cdot n^5$ a exponenciální algoritmus složitosti $2^{n^{1/3}}$. V tomto případě je polynomiální algoritmus efektivnější až pro vstupy délky $n > 10^6$. Pro praktické účely tedy může být algoritmus s uvedenou exponenciální složitostí výhodnější.

Třída NP obsahuje rozhodovací úlohy, pro které nemusí být znám efektivní algoritmus, ale pro které existuje možnost doložit kladnou odpověď efektivně ověřitelným důkazem. Typickými reprezentanty této třídy je problém splnitelnosti Booleovských formulí a úloha rozhodnout, zda daný graf obsahuje kliku velikosti alespoň k, kde číslo k je součástí vstupu. Třída PSPACE je třídou úloh, které lze řešit v polynomiálním prostoru. Příklady úloh, které patří do PSPACE, jsou problém pravdivosti kvantifikovaných booleovských formulí a problémy pravdivosti formulí v některých modálních logikách.

Zařazováním úloh do tříd a pomocí pojmu úplnost ve třídě lze úlohy klasifikovat. Tato klasifikace neposkytne dokazatelnou informaci o složitosti úlohy, ale umožňuje dát složitost dosud neprozkoumané úlohy do vztahu ke složitosti úloh, které již podrobně zkoumány byly. Tento postup je používán nejen pro teoretické účely, ale i jako pomocný prostředek pro hledání efektivních algoritmů.

V této přednášce vyložíme základní pojmy a metody, které se používají ke studiu složitosti obecně a ke klasifikaci úloh pomocí úplnosti ve třídách. Ukážeme také základní modely pro měření složitosti Booleovských funkcí. Kromě obvodů a obecných formulí to jsou především rozhodovací stromy, CNF a DNF (konjunktivní a disjunktivní normální forma).

1.1 Některé obecné pojmy

Úlohou obvykle nebudeme rozumět jedno konkrétní zadání, ale celou množinu všech možných zadání této úlohy. Pokud budeme mluvit o jednom konkrétním zadání, budeme mluvit o instanci úlohy.

Budeme rozlišovat úlohy tří hlavních typů: vyhledávací úloha, výpočet funkce a rozhodovací úloha. Vyhledávací úloha je úloha, kdy hledáme libovolný objekt, který splňuje nějakou vlastnost, například libovolné splňující ohodnocení dané výrokové formule. Výpočet funkce je úloha, která má jednoznačně daný výsledek. Rozhodovací úlohy jsou speciálním případem výpočtu funkce, jejichž hodnotou je odpověď na nějakou otázku typu ANO/NE.

Problém maximální kliky v grafu lze formulovat různými způsoby, které patří k různým typům úloh. Pokud chceme znát některou maximální kliku, jde o vyhledávací úlohu, protože maximálních klik může být v grafu více a kterákoli z nich je správnou odpovědí. Pokud chceme zjistit velikost maximální kliky, jde o výpočet funkce, protože její hodnota je jednoznačně určena. Nejčastěji budeme

problém kliky uvažovat jako rozhodovací úlohu. V tomto případě je součástí vstupu kromě grafu ještě přirozené číslo k a ptáme se, zda graf obsahuje kliku velikosti alespoň k.

Problém maximální kliky patří mezi optimalizační úlohy. To jsou úlohy, jejichž každá instance určuje množinu objektů, které nazýváme přípustná řešení, a na této množině minimalizujeme nebo maximalizujeme nějakou kriteriální funkci. V případě problému maximální kliky je instance úlohy určena konečným grafem. Množina přípustných řešení pro daný graf jsou všechny kliky v tomto grafu a kriteriální funkce je velikost kliky.

1.2 Příklady úloh

1.2.1 Algoritmicky nerozhodnutelné problémy

Aby mělo smysl mluvit o složitosti úlohy, musí být úloha algoritmicky řešitelná. Pro úplnost si uvedeme několik příkladů úloh, které nejsou algoritmicky řešitelné. Následující úlohy jsou částečně rekurzivní, ale nejsou rekurzivní.

Problém zastavení.
Pro libovolný algoritmus a libovolný vstup se má rozhodnout, zda daný algoritmus se pro daný vstup zastaví nebo nikoli.

Dokazatelnost formulí v PA.
Peanova aritmetika je teorií prvního řádu se speciálními symboly pro sčítání a násobení. Úlohou je, pro libovolnou správně utvořenou formuli v jazyce PA zjistit, zda je dokazatelná z axiomů.

Rovnost bezkontextového jazyka množině všech slov.
Nechť Σ je alespoň dvouprvková abeceda. Úlohou je, pro libovolnou bezkontextovou gramatiku G s terminální abecedou Σ zjistit, zda platí $L(G) \neq \Sigma^*$.

Neprázdnost průniku dvou bezkontextových jazyků.
Nechť Σ je alespoň dvouprvková abeceda. Úlohou je, pro libovolné dvě bezkontextové gramatiky G_1 a G_2 s terminální abecedou Σ zjistit, zda je průnik $L(G_1) \cap L(G_2)$ neprázdný.

Řešitelnost polynomu více proměnných s celočíselnými koeficienty v celých číslech.
Úlohou je, pro libovolný polynom $p(x_1, \ldots, x_n)$ více proměnných s celočíselnými koeficienty zjistit, zda existují celá čísla a_i pro $i = 1, \ldots, n$ tak, že $p(a_1, \ldots, a_n) = 0$.

1.2.2 Dokazatelně složité problémy

Pro následující úlohy lze pomocí metody diagonalizace dokázat, že mají vysokou výpočetní složitost. Tím se tyto úlohy liší od úloh z tříd NP a PSPACE, pro které není známo, zda lze pomocí diagonalizace nebo jiným způsobem dokázat

exponenciální dolní odhady jejich složitosti.

Problém zastavení v daném čase.
Uvažujme libovolnou enumeraci M_α Turingových strojů se vstupní abecedou $\{0,1\}$ s libovolným konečným počtem pásek, kde kód α je slovo z bezprefixové množiny slov tvaru $(00+01)^*1$. Nechť $f : \mathrm{N} \to \mathrm{N}$ je rekurzivní funkce. Problémem zastavení v čase f budeme rozumět následující úlohu. Pro libovolný vstup αw, kde α je kód Turingova stroje ve zvoleném kódování a $w \in \{0,1\}^*$, rozhodnout, zda se výpočet M_α pro vstup w zastaví nejvýše po $f(|w|)$ krocích.

Věta 1.2.1 *Nechť $f(n)$ je některá z funkcí 2^n, 2^{2^n}, $2^{2^{2^n}}$, ... Pak každý Turingův stroj pro problém zastavení v čase f provede pro nekonečně mnoho vstupů tvaru $\alpha\alpha$ alespoň $f(|\alpha|) - O(|\alpha|)$ kroků.*

Důkaz: Pro libovolný kód α a vstup w označme jako $T(\alpha, w)$ počet kroků, které provede výpočet M_α pro vstup w, nebo nekonečno, pokud se výpočet nezastaví. Pro libovolnou funkci f uvažovaného tvaru lze sestrojit kód β tak, že M_β se pro libovolný vstup w zastaví právě po $f(|w|)$ krocích, tedy $T(\beta, w) = f(|w|)$. Nechť γ je kód libovolného TS pro problém zastavení v čase f. Zkonstruujeme kód δ tak, že M_δ obsahuje pásky pro simulaci strojů M_β a M_γ a jeho výpočet probíhá následovně. Pro vstup α nejprve M_δ v čase $O(|\alpha|)$ zkopíruje slovo α na vstupní pásku M_β a slovo $\alpha\alpha$ na vstupní pásku M_γ a pak paralelně simuluje

- M_β pro vstup α
- M_γ pro vstup $\alpha\alpha$

tak, že výpočet je ukončen nejpozději, když skončí výpočet M_β, a je ukončen dříve, pokud výpočet M_γ skončí dříve než M_β s výsledkem "nezastaví". Platí tedy

$$T(\delta, \alpha) \leq f(|\alpha|) + O(|\alpha|) \tag{1}$$

a pokud $T(\gamma, \alpha\alpha) < f(|\alpha|)$, pak platí implikace

$$T(\alpha, \alpha) \leq f(|\alpha|) \quad \Rightarrow \quad T(\delta, \alpha) > f(|\alpha|) \tag{2}$$
$$T(\alpha, \alpha) > f(|\alpha|) \quad \Rightarrow \quad T(\delta, \alpha) \leq T(\gamma, \alpha\alpha) + O(|\alpha|)\ . \tag{3}$$

Dokažme, že platí

$$T(\gamma, \delta\delta) \geq f(|\delta|) - O(|\delta|)\ , \tag{4}$$

což je tvrzení věty pro M_γ a vstup $\alpha = \delta$. Pokud $T(\gamma, \delta\delta) \geq f(|\delta|)$, pak (4) platí. Pokud $T(\gamma, \delta\delta) < f(|\delta|)$, pak (2) pro $\alpha = \delta$ vede ke sporu, pokud by platilo $T(\delta, \delta) \leq f(|\delta|)$, a platí tedy $T(\delta, \delta) > f(|\delta|)$. Pak z (3) pro $\alpha = \delta$ plyne

$$f(|\delta|) < T(\delta, \delta) \leq T(\gamma, \delta\delta) + O(|\delta|)\ ,$$

což po úpravě implikuje (4).

Protože lze stroj M_δ zkonstruovat nekonečně mnoha různými způsoby, platí tvrzení věty pro nekonečně mnoho slov α. □

Pravdivost uzavřených formulí v Presburgerově aritmetice
Presburgerova aritmetika se od Peanovy aritmetiky liší tím, že nemá symbol pro násobení. Takto oslabená teorie je rozhodnutelná, ale rozhodnutí o pravdivosti formule délky n vyžaduje v obecném případě $2^{2^{cn}}$ kroků, kde $c > 0$ je konstanta.

Pravdivost uzavřených formulí v aritmetice reálných čísel
Aritmetika reálných čísel omezená na operace sčítání a násobení je rozhodnutelná. Vyplývá z toho například, že je rozhodnutelná rovinná geometrie, pokud se omezíme na konstrukce pravítkem a kružítkem. Zjištění pravdivosti formule délky n ale v obecném případě vyžaduje 2^{cn} kroků, kde $c > 0$ je konstanta.

1.2.3 Pravděpodobně složité problémy

Úlohy uvedené v tomto paragrafu jsou řešitelné úplným výčtem, který vyžaduje exponenciální počet kroků. Jsou to tedy úlohy algoritmicky řešitelné. Pravděpodobně ale neexistuje algoritmus, který by všechny instance kterékoli z těchto úloh řešil v polynomiálním čase.

Problém tautologičnosti Booleovských formulí.
Pro formuli v disjunktivní normální formě rozhodnout, zda je tautologií.

Splnitelnost Booleovské formule (SAT).
Pro libovolnou Booleovskou formuli v konjunktivní normální formě rozhodnout, zda je splnitelná, tedy zda existuje ohodnocení proměnných, které formuli splňuje. Problém SAT je duální k problému tautologičnosti, protože formule je tautologií právě tehdy, když její negace není splnitelná.

Omezená splnitelnost monotonní Booleovské formule.
Pro libovolnou monotonní Booleovskou formuli v konjunktivní normální formě a přirozené číslo k rozhodnout, zda existuje splňující ohodnocení proměnných, ve kterém je nejvýše k jedniček.

Splnitelnost Booleovského obvodu (CSAT).
Pro libovolný Booleovský obvod rozhodnout, zda existuje ohodnocení vstupních proměnných, pro které vydá obvod hodnotu 1.

Problém maximální kliky v grafu.
Klikou v grafu se rozumí množina jeho vrcholů, ve které jsou každé dva vrcholy spojeny hranou. Úloha je, v daném grafu najít kliku maximální velikosti.

Vrcholové pokrytí.
Pro daný graf najít množinu vrcholů minimální velikosti, která má neprázdný průnik s každou hranou grafu.

Problém batohu.
Pro libovolnou $n+1$-tici přirozených čísel $a_1, \ldots, a_n, b$ zjistit, zda existuje množina indexů $I \subseteq \{1, \ldots, n\}$ tak, že $\sum_{i \in I} a_i = b$. Délkou vstupu se pro tuto úlohu rozumí součet délek zápisů vstupních čísel v binárním zápisu.

Hamiltonovská kružnice.
Pro libovolný neorientovaný graf zjistit, zda v něm existuje uzavřená cesta, která prochází každým vrcholem právě jednou.

Problém obchodního cestujícího.
Pro libovolný graf s hranami ohodnocenými kladnými čísly nalézt v grafu uzavřenou cestu, která prochází každým vrcholem právě jednou a jejíž cena (součet ohodnocení hran) je minimální.

1.2.4 Složitost některých jednoduchých úloh

Násobení matic.
Kromě standardního algoritmu, který násobí dvě matice $n \times n$ v čase n^3, existuje Strassenův algoritmus, který tyto matice vynásobí v čase $C\, n^{\log_2 7}$, což je přibližně $C\, n^{2.81}$. Existuje také asymptoticky rychlejší algoritmus, který pracuje v čase nejvýše $Cn^{2.3728639}$, ale konstanta C je v tomto případě tak veliká, že algoritmus získává výhodu proti obvyklým algoritmům až pro astronomické hodnoty n.

Násobení přirozených čísel
Základní algoritmus pro násobení čísel, jejichž binární zápis má délku n bitů, vyžaduje $C\, n^2$ kroků. Existuje algoritmus, který využívá diskrétní Fourierovu transformaci a pracuje v čase $C\, n \log n \log\log n$. Tento algoritmus je využíván v matematickém software pro velká čísla, například když $n > 10^5$.

1.3 Značení

Pro libovolnou nezápornou funkci f znamená

- $O(f)$ libovolnou nezápornou funkci g, pro kterou existuje n_0 a konstanta c tak, že $(\forall n \geq n_0)\ g(n) \leq cf(n)$.
- $\Omega(f)$ libovolnou nezápornou funkci g, pro kterou existuje n_0 a konstanta $\varepsilon > 0$ tak, že $(\forall n \geq n_0)\ g(n) \geq \varepsilon f(n)$.
- $\Theta(f)$ libovolnou nezápornou funkci g, která je současně $O(f)$ i $\Omega(f)$.

Speciálními případy tohoto značení je $O(1)$, $\Omega(1)$ a $\Theta(1)$, které v uvedeném pořadí znamenají funkce, které jsou shora, zdola a z obou stran omezené kladnou konstantou.

1.4 Měření složitosti

Složitost algoritmu obvykle roste s velikostí vstupního zadání a může být různá pro různé instance stejné velikosti. Pro jednoduchost se nejčastěji uvažuje složitost v nejhorším případě, která je vyjádřena funkcí, která závisí na velikosti

vstupu a pro danou velikost vstupu je rovna maximu ze složitosti řešení přes všechny instance dané velikosti.

Složitosti algoritmů budeme porovnávat asymptoticky, což znamená, že algoritmus se složitostí $t_1(n)$ považujeme za rychlejší než algoritmus se složitostí $t_2(n)$, pokud existuje n_0 tak, že pro všechna $n \geq n_0$ je $t_1(n) < t_2(n)$. Porovnávání složitosti tedy závisí pouze na složitosti pro vstupy, jejichž velikost konverguje do nekonečna.

Nelze exaktně definovat pojem miminální složitost algoritmu pro řešení dané úlohy. Existují úlohy, pro které ke každému algoritmu existuje algoritmus, který je asymptoticky exponenciálně rychlejší. Z tohoto důvodu nebudeme definovat pojem složitosti úlohy, ale pouze budeme mluvit o tom, zda je úloha řešitelná v čase $O(t(n))$ pro nějakou danou mez $t(n)$ nebo v prostoru $O(s(n))$ pro nějakou funkci $s(n)$.

1.5 Použité vzorce

Nechť n je přirozené číslo a označme jako l délku jeho binárního zápisu. Pak $2^{l-1} \leq n \leq 2^l - 1$, tedy $2^{l-1} < n + 1 \leq 2^l$. Z toho plyne $l - 1 < \log_2(n+1) \leq l$, a tedy

$$l = \lceil \log_2(n+1) \rceil \ .$$

Funkce e^x je konvexní, funkce $1 + x$ je lineární a tyto funkce mají v bodě $x = 0$ stejnou hodnotu i první derivaci. Pro každé reálné x tedy platí

$$\begin{aligned} 1 + x &\leq \mathrm{e}^x \\ 1 - x &\leq \mathrm{e}^{-x} \end{aligned}$$

a pro nenulové x platí

$$\begin{aligned} 1 + x &< \mathrm{e}^x \\ 1 - x &< \mathrm{e}^{-x} \ . \end{aligned}$$

2 Zavedení základních pojmů

2.1 Úlohy

V této kapitole popíšeme, jak budeme reprezentovat úlohy pro účely teorie složitosti. Později se budeme zabývat především rozhodovacími úlohami, ale v této úvodní kapitole se budeme věnovat i dalším typům úloh. Ukážeme, že z hlediska složitosti není omezení na rozhodovací úlohy na úkor obecnosti.

2.1.1 Reprezentace úloh pomocí jazyků, efektivní kódování

Úloha pro účely teorie výpočtové složitosti se skládá z popisu vstupu a požadovaného výstupu. Vstup i výstup lze obvykle chápat jako jeden nebo několik matematických objektů. Uváděli jsme si již úlohy, ve kterých byl vstupem graf případně s nějakými dalšími údaji, čísla, koeficienty polynomu, bezkontextové gramatiky a pod. Výstupem může být odpověď na nějakou otázku nebo opět nějaký objekt, např. cesta v grafu, podmnožina vrcholů grafu a pod.

Pro účely zkoumání tříd úloh je třeba úlohy reprezentovat jednotným způsobem. Z hlediska řešení úloh na TS je přirozené reprezentovat vstup i výstup výpočtu pomocí posloupností symbolů v konečných abecedách. Jestliže Σ je konečná abeceda, budeme libovolnou posloupnost symbolů ze Σ nazývat slovo nad abecedou Σ. Množinu všech slov nad Σ budeme značit Σ^*.

Úlohy, které počítají obecnou funkci, budeme reprezentovat funkcí ve tvaru

$$f : \Sigma_1^* \to \Sigma_2^*, \tag{5}$$

kde Σ_1 a Σ_2 jsou konečné abecedy. To znamená, že pro úlohy, jejichž vstup není posloupností znaků, musíme vždy určit vhodnou abecedu a způsob, jak instance dané úlohy kódovat pomocí posloupností znaků ve zvolené abecedě. Při kódování instancí pomocí posloupností znaků obvykle jen některé posloupnosti kódují instance. Na posloupnostech, které nekódují instanci je funkce (5) nedefinována nebo nabývá speciální hodnotu, která označuje, že vstupní posloupnost není kódem instance.

Rozhodovací úlohy budeme reprezentovat pomocí jazyků.

Definice 2.1.1 Jazyk nad konečnou neprázdnou abecedou Σ je libovolná podmnožina $L \subseteq \Sigma^*$.

Jazyk L reprezentující úlohu je množina slov, které jsou kódem instance, na které dává úloha kladnou odpověď. Slova, která nekódují instanci, a slova, která kódují instanci se zápornou odpovědí, definice jazyka L nerozlišuje.

Pokud mluvíme o konkrétní úloze, popisujeme obvykle vstup a výstup jako matematické objekty, tj. pomocí běžných matematických pojmů, bez toho, že bychom přesně specifikovali kódování instancí pomocí posloupností symbolů. Když budeme mluvit o třídách úloh, budeme vždy mluvit o třídách jazyků. Zařazení nějaké úlohy do třídy pak znamená, že do této třídy patří jazyk, který úlohu reprezentuje. Tato reprezentace není jednoznačně určena. Přesto není nutné vždy reprezentaci přesně specifikovat, protože obvykle lze nějaké kódování pro danou úlohu navrhnout a navíc všechna "rozumná" kódování vedou na algoritmy, jejichž složitost se liší nejvýše polynomiálně.

Lze zkonstruovat umělá kódování, při kterých je i jednoduchý problém obtížný díky tomu, že je obtížné rekonstruovat původní instanci z jejího kódu. Naopak, pokud kód instance prodloužíme na neúměrně velkou délku, může být složitost úlohy vyjádřena neopodstatněně malou funkcí délky vstupu.

Například problém batohu se stane řešitelný v polynomiálním čase, pokud čísla v jeho instancích vyjádříme v jedničkové soustavě. Toto je ovšem podstatná změna kódování, která v podstatě definuje jinou úlohu. Abychom se vyhnuli tomuto typu problému, požadujeme, aby čísla byla kódována binárně.

Protože nelze přesně specifikovat, co je to rozumná reprezentace, zformulujeme jen velmi obecnou podmínku, kterou by mělo kódování splňovat. Vycházíme z toho, že pro každou úlohu existuje nějaká "přirozená" reprezentace vstupu pomocí matematických objektů. Požadujeme, aby kódování těchto objektů pomocí posloupností symbolů bylo takové, že transformaci mezi touto posloupností a přirozenou reprezentací vstupu lze provést oběma směry v polynomiálním čase. Tato definice je ovšem pouze intuitivní, protože se odkazuje na pojem obecného matematického objektu, který nemáme přesně definován.

2.1.2 Porovnání obecných a rozhodovacích úloh

Nyní ukážeme, že z hlediska rozlišování polynomiální a nepolynomiální složitosti je možné se omezit jen na rozhodovací úlohy. Pokud uvažujeme o výpočtu funkcí v polynomiálním čase, musí být výsledek funkce vyjádřitelný slovem polynomiální délky. Ke každé úloze na výpočet funkce, která má tuto vlastnost, nalezneme rozhodovací úlohu, která je z hlediska existence polynomiálního algoritmu ekvivalentní původní úloze. Nejprve ukážeme přirozené příklady rozhodovacích úloh, které jsou odvozeny z úloh obecnějších, a nemohou být řešitelné polynomiálním algoritmem, pokud původní úloha není takto řešitelná.

Příklad. Algoritmus pro zjišťování existence splňujícího ohodnocení Booleovské formule lze použít na nalezení lexikograficky minimálního splňujícího ohodnocení.

Postup je následující. Je-li formule $\varphi(x_1, \ldots, x_n)$ splnitelná, použijeme test splnitelnosti na formule $\varphi(0, x_2, \ldots, x_n)$ a $\varphi(1, x_2, \ldots, x_n)$. Alespoň jedna z těchto formulí je splnitelná. Pokud jsou splnitelné obě, vybereme formuli s dosazením 0. V každém případě získáme splnitelnou formuli, na kterou použijeme rekurzivně stejný postup. Tímto způsobem nalezneme lexikograficky minimální splňující ohodnocení.

Příklad. Algoritmus pro zjišťování existence Hamiltonovské kružnice lze použít na nalezení některé z nich. Opět bychom mohli hledat kružnici, která splňuje nějakou zjednoznačňující podmínku, ale pro jednoduchost to nebudeme dělat.

Postup je následující. Jestliže graf obsahuje Hamiltonovskou kružnici, probíráme jeho hrany v nějakém pořadí a odstraníme každou hranu, jejíž odstranění vede opět na graf obsahující Hamiltonovskou kružnici. Po probrání všech hran obsahuje graf pouze hrany, které tvoří některou z Hamiltonovských kružnic původního grafu.

Důkaz provedeme sporem. Postup zaručuje, že výsledný graf obsahuje alespoň jednu Hamiltonovskou kružnici. Kdyby obsahoval ještě nějakou další hranu, pak odstranění této hrany vede na graf s Hamiltonovskou kružnicí. Tato hrana tedy musela být odstraněna v předchozích krocích algoritmu.

Jako další příklad si uveďme optimalizační úlohy. Optimalizační úlohy lze snadno převést na rozhodovací úlohu tím, že do vstupu úlohy doplníme další parametr, řekněme b. Úlohou pak je, zjistit, zda mezi přípustnými řešeními existuje takové, na němž má kriteriální funkce hodnotu aspoň (nejvýše) b. Řešení původní úlohy lze získat opakovaným voláním uvedeného rozhodovacího problému metodou půlení intervalu.

Následující věta ukazuje zobecnění uvedených tří konstrukcí pro libovolnou úlohu na výpočet funkce.

Věta 2.1.2 *Nechť $f : \Sigma_1^* \to \Sigma_2^*$ je funkce, pro kterou je $|f(w)|$ omezeno polynomem od $|w|$. Nechť $\# \notin \Sigma_1 \cup \Sigma_2$ je pomocný symbol a nechť $L_f = \{w\#u \,;(\exists v)uv = f(w)\}$. Pak je jazyk L_f rozpoznatelný na TS v polynomiálním čase právě tehdy, když je funkce $f(w)$ vyčíslitelná v polynomiálním čase.*

Důkaz: Pokud je funkce $f(w)$ vyčíslitelná v polynomiálním čase, pak lze pro libovolné slovo $w\#u$ rozhodnout, zda patří do L_f tím, že vypočteme $f(w)$ a porovnáme u s počátečním úsekem $f(w)$ stejné délky.

Algoritmus pro výpočet $f(w)$ s pomocí testování podmínek typu $w\#u \in L_f$ je následující. Postupně konstruujeme prodlužující se počáteční úseky slova $f(w)$ počínaje prázdným slovem. Jestliže v nějakém kroku je doposud nalezený úsek u, pak se pro všechny symboly $a \in \Sigma_2$ zjistí, zda slovo $w\#ua$ patří do L_f. Toto se může stát nejvýše pro jeden symbol a. Pokud to nastane, pak ua je nový počáteční úsek $f(w)$. Pokud žádný symbol a nesplňuje uvedenou vlastnost, je $f(w) = u$.

Tento algoritmus vyžaduje nejvýše $(|f(w)| + 1)|\Sigma_2|$ testů, zda $w\#u \in L_f$ pro některá slova v délky nejvýše $|f(w)| + 1$. Nechť otázku, zda $w\#u \in L_f$ lze rozhodnout v čase $p(|w\#u|)$, kde p je neklesající polynom. Nechť $|f(w)| \leq g(|w|)$, kde g je polynom. Pak popsaný výpočet vyžaduje nejvýše

$$|\Sigma_2|\ (g(|w|) + 1)\ p(|w| + 1 + g(|w|) + 1)$$

kroků. Vzhledem k tomu, že součet, součin a složení polynomů je polynom, je věta dokázána. □

2.2 Výpočetní modely

Jako základní výpočetní model pro teoretické zkoumání použijeme Turingův stroj. Důvodem pro to je, že tento stroj lze velmi jednoduše simulovat pomocí dalších struktur, například pomocí Boleovských obvodů. Tuto simulaci později použijeme k důkazu existence NP-úplné úlohy. Pro porovnání složitosti TS a běžných počítačů pak ještě použijeme RAM (random access machine).

2.2.1 Turingův stroj

Jako výpočetní model pro řešení úloh použijeme Turingův stroj. Budeme uvažovat jednopáskový a vícepáskový TS. Vícepáskový stroj má proti jednopáskovému výhodu například při kopírování delšího úseku pásky. Jednopáskový stroj musí opakovaně navštěvovat výchozí a cílové místo kopírování, protože může přenášet jen omezený počet symbolů při jednom přechodu. Vícepáskový stroj může kopírovanou část zapsat na pomocnou pásku a z ní pak přenést na nové místo. Z tohoto důvodu se jako standardní model pro definici výpočtové složitosti používá vícepáskový stroj. V některých důkazech je ale výhodnější pracovat s jednopáskovým strojem pro jeho jednoduchost, a proto popíšeme oba tyto stroje.

Jednopáskový TS se skládá z jednostranně nekonečné pracovní pásky, řídící jednotky a čtecí hlavy na pásce. Páska se skládá z buněk (polí), z nichž každá může obsahovat symbol pracovní abecedy Γ. Abeceda Γ obsahuje prázdný symbol ε.

Vstupní slovo je slovo v abecedě $\Sigma \subseteq \Gamma \setminus \{\varepsilon\}$. Na počátku výpočtu TS předpokládáme, že vstupní slovo pro je zapsáno v počátečním úseku pásky a že všechny ostatní buňky pásky obsahují prázdný symbol ε. Pokud TS počítá

funkci, předpokládáme, že po skončení výpočtu je vypočtená hodnota zapsána v počátečním úseku pásky a zbytek pásky je prázdný. Pokud TS řeší rozhodovací úlohu, není výstup zapsán na pásku a výstup je určen koncovým stavem.

Vícepáskový stroj obsahuje vstupní pásku, která je určena pouze pro čtení, libovolný počet pracovních pásek a pokud TS počítá funkci, může obsahovat výstupní pásku, která je určena pouze pro zápis výsledku. Každá z pásek TS se skládá z jednostranně nekonečné posloupnosti buněk. Připouštíme, že různé pásky používají znaky různých abeced. Každá buňka každé pásky obsahuje jeden znak příslušné abecedy, který může být prázdný. Pracovní abecedy na páskách budeme značit Γ, případně s indexy. Na počátku výpočtu předpokládáme, že všechny buňky obsahují prázdný symbol, kromě počátečního úseku vstupní pásky, který obsahuje vstupní slovo.

V některých případech budeme uvažovat rozdělení pásky na stopy. Pokud uvažujeme k stop, znamená to, že každé pole pásky obsahuje k-tici symbolů z abecedy $\Gamma_1 \times \ldots \times \Gamma_k$, kde Γ_i je abeceda i-té stopy. Turingův stroj čte všechny tyto symboly současně, ale při zápisu programu pro Turingův stroj je budeme rozlišovat.

Hlavy na páskách mohou číst symbol na pozici, kde se právě nachází a mohou tento symbol přepsat jiným symbolem. Kromě toho se může hlava posunout o jednu buňku vlevo nebo vpravo nebo zůstat na místě. Pokud se stroj pokusí provést krok vlevo na nejlevější buňce pásky, skončí výpočet chybou.

Činnost hlav na páskách je řízena řídící jednotkou, která se může nacházet v konečně mnoha stavech. Množinu stavů řídící jednotky budeme značit Q. Činnost řídící jednotky je popsána přechodovou funkcí. Jednopáskový stroj s množinou stavů Q a abecedou na pásce Γ má přechodovou funkci

$$f : Q \times \Gamma \to Q \times \Gamma \times \{-1, 0, 1\} \ ,$$

která je obvykle pro některé vstupy nedefinována a tato situace znamená ukončení výpočtu. Přechodová funkce je interpretována tak, že $f(q, a) = (q', a', s)$ znamená, že je-li výchozí stav q a symbol čtený na pásce je a, bude zapsán na pásku symbol a', řídící jednotka přejde do stavu $q' \in Q$ a hlava se posune o s polí vpravo, tedy ve skutečnosti vlevo, pokud $s = -1$. Pokud je čtené pole pásky prázdné, je jako argument přechodové funkce interpretováno stejně jako pole obsahující $\varepsilon \in \Gamma$. Prázdný symbol ε může být zapsán na pásku, ale každé pole, do kterého byl zapsán symbol, již zůstává neprázdné po celý zbytek výpočtu. Úsek pásky se symboly z abecedy Γ tak přesně určuje, která pole již byla při výpočtu navštívena.

Přechodová funkce vícepáskového TS s k páskami s abecedami $\Gamma_1, \ldots, \Gamma_k$ je tvaru

$$f : Q \times \Gamma_1 \times \ldots \Gamma_k \to Q \times \Gamma_1 \times \ldots \Gamma_k \times \{-1, 0, 1\}^k \ .$$

Jestliže $f(q, (a_1, \ldots a_k)) = (q', (a'_1, \ldots a'_k), (s_1, \ldots s_k))$, je stav $q \in Q$ opět výchozí stav, jednotlivé symboly $a_i \in \Gamma_i$ jsou symboly čtené na jednotlivých páskách, stav $q' \in Q$ je stav, do kterého řídící jednotka přejde po provedení instrukce, symboly $a'_i \in \Gamma_i$ jsou symboly, které budou zapsány na jednotlivé pásky a čísla s_i zaznamenávají pro každou pásku zvlášť, jak se má posunout čtecí hlava.

Pro speciální účely se používá také nedeterministický TS, pro který je hodnotou přechodové funkce obecně množina několika možných akcí. V tomto případě může výpočet pokračovat kteroukoli z těchto přípustných akcí. Takovýto TS se používá pro rozpoznávání jazyka a může mít pro jeden vstup více možných výpočtů. Vstupní slovo je přijato, pokud alesoň jeden z možných výpočtů je přijímající. Nedeterministický TS využívá jedna možných definic třídy NP, ale nebudeme tuto definici používat.

Při výpočtu TS závisí v každé situaci další krok na stavu řídící jednotky a symbolech čtených na páskách. Množina stavů Q obsahuje jeden nebo dva koncové stavy. Pro koncový stav není přechodová funkce definována. Pro výpočet funkce stačí jeden koncový stav q_+ a jestliže do něj TS přejde, je výpočet ukončen jako korektní. Pokud TS dojde do stavu, ve kterém není přechodová funkce definována, ale není to stav q_+ výpočet končí chybou. Pokud TS rozpoznává jazyk, jsou koncové stavy q_+ a q_-. Stav q_+ znamená přijetí slova a všechny ostatní stavy, kde je přechodová funkce nedefinována včetně q_-, znamenají zamítnutí vstupního slova. Jazyk rozpoznávaný takovým TS je množina všech slov, které jsou strojem přijaty.

Časová a prostorová míra složitosti pro TS jsou definovány následovně.

Definice 2.2.1 Pro daný TS M a libovolné vstupní slovo w nechť $t_M(w)$ označuje počet kroků nutných k výpočtu M pro w. Časovou složitostí M pak rozumíme funkci $t_M(n)$ definovanou jako

$$t_M(n) = \max\{t_M(w); |w| = n\}$$

Definice 2.2.2 Pro daný TS M a libovolné vstupní slovo w nechť $s_M(w)$ označuje maximum počtu polí navštívených na jednotlivých pracovních páskách během výpočtu M pro w. Prostorovou složitostí M pak rozumíme funkci $s_M(n)$ definovanou jako

$$s_M(n) = \max\{s_M(w); |w| = n\}$$

2.2.2 Random access machine

RAM (random access machine) je model počítače, jehož struktura připomíná strukturu běžných počítačů. Ukážeme, že RAM lze simulovat pomocí TS v polynomiálním čase a že TS lze simulovat pomocí RAM v lineárním čase. To dokazuje, že množina úloh, které jsou řešitelné v polynomiálním čase na RAM a na TS jsou shodné.

Podobně jako TS se i RAM skládá z řídící jednotky řízené programem a z datové struktury, jejíž obsah je během výpočtu měněn. Datová strutura RAM se skládá z nekonečně mnoha registrů, z nichž každý může uložit libovolně velké celé číslo. Počáteční hodnota všech registrů je 0. Registry jsou očíslovány a obsah i-tého registru pro $i = 0, 1, 2, \ldots$ budeme značit r_i. Čísla registrů odpovídají adresám v operační paměti počítače. Kromě registrů má RAM přístup ke vstupní posloupnosti čísel $(v_1, v_2, \ldots, v_n)$. Ze vstupu do registrů a mezi registry lze provádět přesuny dat. Lze provádět základní aritmetické operace mezi libovolnými registry a výsledek zapsat do libovolného registru. Lze testovat, zda hodnota registru je kladná, nulová nebo záporná. Lze provést podmíněný skok

na základě vyhodnocení některého z výše uvedených testů a lze provést také nepodmíněný skok.

Program pro RAM je posloupnost instrukcí $\pi_1, \pi_2, \ldots, \pi_m$. Řídící jednotka provádí vždy tu instrukci, jejíž pořadové číslo je uloženo v čítači instrukcí κ, tj. instrukci π_κ. Počáteční hodnota κ je 1. Většina instrukcí zvětší κ o jedničku. Instrukce skoků mohou nastavit κ na libovolnou hodnotu danou programem.

Přesný seznam instrukcí a jejich význam jsou dány následujícími tabulkami. Aritmetické instrukce mají tři parametry, které určují adresy dvou vstupních hodnot a adresu, kam má být zapsán výsledek. Budeme uvažovat následující typy parametrů instrukcí:

název	**parametr**	**hodnota, se kterou se operace provede**
konstanta	$=i$	číslo i
přímá adresace	i	r_i
nepřímá adresace	$*i$	r_{r_i}

Parametry instrukcí těchto tří typů budeme značit α, β, γ. Budeme uvažovat ještě další typy parametrů, které odkazují do vstupní posloupnosti a mohou být těchto typů.

název	**parametr**	**hodnota, se kterou se operace provede**
přímá adresace	i	v_i
nepřímá adresace	$*i$	v_{r_i}

Parametr operace READ, který může být libovolného z těchto dvou typů, budeme značit δ.

Instrukce	**Operand**	**Význam**
READ	δ, β	$\beta \leftarrow \delta$
COPY	α, β	$\beta \leftarrow \alpha$
ADD	α, β, γ	$\gamma \leftarrow \alpha + \beta$
SUB	α, β, γ	$\gamma \leftarrow \alpha - \beta$
MUL	α, β, γ	$\gamma \leftarrow \alpha * \beta$
DIV	α, β, γ	$\gamma \leftarrow \alpha/\beta$ (zaokrouhleno vždy k nule)
JUMP	$=i$	$\kappa \leftarrow i$
JPOS	$\alpha, =i$	`if` $\alpha > 0$ `then` $\kappa \leftarrow i$
JZERO	$\alpha, =i$	`if` $\alpha = 0$ `then` $\kappa \leftarrow i$
JNEG	$\alpha, =i$	`if` $\alpha < 0$ `then` $\kappa \leftarrow i$
RETURN	$=i$	zastavit výpočet a vydat hodnotu i

Obrázek 1: Instrukce random access machine.

Seznam instrukcí RAM je na Obrázku 1. Pokud instrukce nemění κ explicitně, znamená to, že se κ nahradí $\kappa + 1$. Pokud některou operaci nelze provést, např. je-li adresa operandu záporná nebo je požadováno dělení nulou, končí výpočet chybou. Při ukončení výpočtu instrukcí RETURN závisí význam vráceného čísla na konkrétním programu.

Budeme uvažovat dva způsoby měření času stroje RAM - jednotkovou a bitovou (logaritmickou) míru. V jednotkové míře se provedení každé instrukce považuje za jeden krok, nezávisle na velikosti čísel, se kterými instrukce pracuje. V bitové míře se za složitost provedení instrukce považuje celkový počet bitů ve dvojkovém zápisu všech čísel zůčastněných na provedení instrukce, tj. včetně adres operandů. Název míry logaritmická je odvozen z toho, že délka binárního zápisu přirozeného čísla je zhruba rovna jeho dvojkovému logaritmu.

Jednotková míra je pohodlnější při odhadování složitosti konkrétních algoritmů. Tato míra je ovšem realistickou mírou složitosti jen pro programy, které při svém běhu pracují s omezeně velkými čísly.

Definice 2.2.3 *Budeme říkat, že RAM pro řešení určité úlohy splňuje polynomální omezení velikosti čísel, jestliže všechna čísla která vzniknou během výpočtu mají absolutní hodnotu shora omezenou polynomem od počtu kroků výpočtu.*

Absolutní hodnota všech čísel použitých ve výpočtu délky t je pak nejvýše $t^{O(1)}$ a délka jejich binárního zápisu je tedy nejvýše $\log t^{O(1)} = O(\log t)$. V takovém případě se jednotková a logaritmická míra složitosti liší jen o logaritmický faktor, což je při polynomiální složitosti algoritmu zanedbatelné. Navíc, v konkrétních aplikacích se v takovém případě obvykle všechna čísla použitá při výpočtu vejdou do slov, s nimiž počítač pracuje, a počet instrukcí RAM je tedy v lineárním vztahu k počtu skutečných instrukcí počítače. Rovnost nemusí platit, protože simulace jedné instrukce jednoho počítače si může vyžádat několik instrukcí druhého počítače.

2.2.3 Booleovský obvod

Booleovský obvod je acyklický graf, jehož vstupní uzly jsou přiřazeny n vstupním proměnným a ostatní uzly jsou ohodnoceny logickými spojkami z množiny, kterou budeme nazývat báze. Báze může být libovolná konečná množina Booleovských funkcí, ale pro jednoduchost budeme jako bázi uvažovat pouze $\{\wedge, \vee, \neg, 0, 1\}$. Některé uzly obvodu jsou určeny jako výstupní. Pokud má obvod n vstupních a m výstupních uzlů, počítá funkci $\{0,1\}^n \to \{0,1\}^m$. Počet hran, které vedou do uzlu, který není vstupní, je roven počtu argumentů spojky, která je uzlu přiřazena. Počet hran, které z uzlu vychází, není omezen. Velikostí obvodu se rozumí počet uzlů, kterým jsou přiřazeny nekonstantní spojky.

Věta 2.2.4 *Libovolnou funkci $f : \{0,1\}^n \to \{0,1\}^m$ lze vyjádřit obvodem velikosti nejvýše $m2^{n+2}$.*

Důkaz: Předpokládejme nejprve $m = 1$ a dokažme indukcí podle n, že pro libovolnou funkci $f : \{0,1\}^n \to \{0,1\}$ existuje obvod velikosti nejvýše $2^{n+2} - 4$. Pro $n = 1$ je $2^{n+2} - 4 = 4$ a tvrzení je tedy zřejmé, protože obvod buď neobsahuje nekonstantní spojku nebo obsahuje jednu negaci. Pokud $n \geq 2$, vyjádříme f ve tvaru

$$f(x_1, x_2, \ldots, x_n) = x_1 \wedge f(1, x_2, \ldots, x_n) \vee \neg x_1 \wedge f(0, x_2, \ldots, x_n) \ .$$

Každou z funkcí $f(1, x_2, \ldots, x_n)$ a $f(0, x_2, \ldots, x_n)$ lze podle indukčního předpokladu vyjádřit obvodem velikosti nejvýše $2^{n+1} - 4$, tedy funkci f lze vyjádřit obvodem velikosti nejvýše $2(2^{n+1} - 4) + 4 = 2^{n+2} - 4$, což dokazuje indukční krok.

Funkci $f : \{0,1\}^n \to \{0,1\}^m$ pro libovolné m lze vyjádřit obvodem, který je tvořem m obvody, z nichž každý počítá jeden výstupní bit. Celková velikost takového obvodu je nejvýše $m2^{n+2}$ a tvrzení věty je tedy dokázáno. □

2.3 Základní třídy složitosti

Třídy jazyků, které popíšeme v následujících dvou definicích, jsou jedním ze základních typů tříd v teorii složitosti. Složitost konkrétních úloh reprezentovaných jako jazyk nad konečnou abecedou lze vyjádřit tím, do které z těchto tříd patří a do kterých ne, kde t nebo s jsou funkce z určité standardizované škály funkcí jako např. $n^k, 2^{cn}, 2^{n^k}$ a pod.

Definice 2.3.1 *Nechť $t : N \to N$ je libovolná funkce. Pak* DTIME(t) *označuje třídu jazyků definovanou následovně. Jazyk L nad libovolnou konečnou abecedou patří do* DTIME(t) *právě tehdy, když existuje TS M, který rozpoznává L, a časová složitost M splňuje $t_M(n) = O(t(n))$.*

Definice 2.3.2 *Nechť $s : N \to N$ je libovolná funkce. Pak* DSPACE(s) *označuje třídu jazyků, která je definována následovně. Jazyk L nad libovolnou konečnou abecedou patří do* DSPACE(s) *právě tehdy, když existuje TS M se vstupní páskou pouze pro čtení, který rozpoznává L, a prostorová složitost M splňuje $s_M(n) = O(s(n))$.*

Třída P reprezentuje úlohy, které jsou řešitelné v čase, který je omezen polynomem od délky vstupu. Formální definice je následující.

Definice 2.3.3

$$\mathrm{P} = \bigcup_{k=1}^{\infty} \mathrm{DTIME}(n^k).$$

Zde využíváme faktu, že pro každý polynom $p(n)$ stupně k platí $p(n) = O(n^k)$. Analogicky budeme definovat třídu PSPACE.

Definice 2.3.4

$$\mathrm{PSPACE} = \bigcup_{k=1}^{\infty} \mathrm{DSPACE}(n^k).$$

2.4 Vzájemné simulace jedno a vícepáskového TS a RAM

K porovnání výpočetní síly modelů se používají vzájemné simulace. Pokud může být libovolný výpočet v jednom modelu simulován výpočtem v jiném modelu bez podstatného nárůstu složitosti, znamená to, že druhý model je alespoň tak silný jako model první. V této kapitole zformulujeme výsledky o simulaci vícepáskového TS pomocí TS s jednou nebo dvěma páskami a dále simulaci RAM pomocí TS a naopak. Většinu tvrzení v této sekci uvádíme bez důkazu. Chybějící důkazy lze nalézt v části Výpočtová složitost II.

Věta 2.4.1 *Každý vícepáskový TS M lze simulovat*
1. TS M' se dvěma pracovními páskami v abecedě $\{0,1\}$,
2. TS M'' s jednou páskou
tak, že výpočet o t krocích je simulován
1. výpočtem M' délky $O(t \log t)$ kroků,
2. výpočtem M'' délky $O(t^2)$ kroků.

Simulaci jednopáskového TS pomocí RAM lze provést následujícím způsobem.

Věta 2.4.2 *Jestliže nějaká úloha je řešitelná na TS v čase t, pak je řešitelná na RAM s jednotkovou mírou v čase $O(t)$. Simulující stroj navíc splňuje podmínku na polynomiální omezení velikosti čísel.*

Důkaz: RAM používá r_0 a r_1 jako pomocné registry a do zbylých registrů ukládá kódy znaků na pásce TS. V registru r_0 je vždy adresa registru, který obsahuje znak čtený hlavou TS. Posun hlavy o jedno pole vlevo nebo vpravo se realizuje zmenšením nebo zvětšením tohoto registu o jedna. V každém kroku simulace RAM do registru r_1 zkopíruje pomocí nepřímého adresování přes r_0 kód znaku čteného čtecí hlavou TS a postupně od něj odečítá jedničky a testuje jej na nulu. Podle toho, po kolika krocích je $r_1 = 0$, rozliší, jaký znak je na pásce zapsán a může podle toho rozvětvit výpočet. Jeden krok TS je realizován shora omezeným počtem instrukcí, a počet nutných kroků je tedy $O(t)$. Registr r_0 je jediný registr, který může nabývat neomezených hodnot a jeho hodnota je nejvýše $t + 2$. Polynomiální omezení velikosti čísel je tedy splněno. □

Pro obecnou simulaci RAM pomocí TS nebudeme nejprve předpokládat žádné omezení velikosti čísel a použijeme tedy bitovou míru složitosti.

Věta 2.4.3 *Jestliže nějaká úloha je řešitelná na RAM s bitovou (logaritmickou) mírou složitosti b, pak je řešitelná na vícepáskovém TS v čase $O(b^2)$.*

Pokud RAM splňuje polynomiální omezení velikosti čísel, můžeme použít jednotkovou míru složitosti a platí následující tvrzení.

Věta 2.4.4 *Jestliže nějaká úloha je řešitelná na RAM s jednotkovou mírou složitosti k a jestliže RAM splňuje polynomiální omezení velikosti čísel, pak je tato úloha řešitelná na vícepáskovém TS v čase $O(k^2 \log k)$.*

2.5 Simulace TS pomocí Booleovských obvodů

Ukážeme, že každý jazyk $L \in \mathrm{P}$ lze rozpoznávat vhodnou posloupností Booleovských obvodů polynomiální velikosti. Jestliže $L \subseteq \Sigma^*$, je potřeba kódovat slova v abecedě Σ pomocí posloupností 0,1, aby mohla být vstupem Booleovského obvodu. Pro konstrukci obvodu, který simuluje TS, budeme uvažovat kódování abecedy Γ, která obsahuje prázdný symbol ε, abecedu Σ a všechny symboly pracovní abecedy TS.

Definice 2.5.1 Nechť Γ je abeceda obsahující znak ε pro prázdný symbol. Kódováním Γ nazveme libovolné prosté zobrazení $k : \Gamma \to \{0,1\}^d$, kde $d \in N$ splňuje $2^d \geq |\Gamma|$. Jestliže $w = a_1 a_2 \dots a_m$, $a_i \in \Gamma$ a $m \leq n$, pak definujme $\mathrm{kod}_n(k, w) = k(a_1) \dots k(a_m) k(\varepsilon)^{n-m}$.

Ukážeme, že pro jazyk L, který lze rozpoznat v polynomiálním čase, lze sestrojit polynomiálně velké obvody, které rozpoznávají slova L ve výše popsaném kódování.

Pro simulaci TS pomocí Booleovských obvodů budeme potřebovat záznam aktuálního stavu výpočtu TS v kterémkoli kroku. Tento záznam budeme nazývat konfigurace a musí obsahovat stav řídící jednotky, obsahy všech pásek a polohy hlav na nich.

Definice 2.5.2 Nechť M je jednopáskový TS s množinou stavů Q a pracovní abecedou Γ. Konfigurace M v prostoru ℓ v některém kroku výpočtu je dvojice slov $x_1 \dots x_\ell$ a $y_1 \dots y_\ell$. Slovo $x_1 \dots x_\ell \in \Gamma^\ell$ je obsah prvních ℓ polí pracovní pásky. Slovo $y_1 \dots y_\ell \in (Q \cup \{\varepsilon\})^\ell$ budeme nazývat stavovým slovem a je určeno následovně. Jestliže se čtecí hlava M nachází na pozici i, pak y_i je stav M v daném kroku výpočtu a $y_j = \varepsilon$ pro $j \neq i$.

Symboly x_i a y_i v konfiguracích budeme kódovat pomocí bitů, přičemž kódování pro x_i bude označeno k_1 a kódování pro y_i bude označeno k_2.

Věta 2.5.3 *Nechť $L \in \mathrm{P}$ je jazyk nad abecedou Σ. Nechť Γ je pracovní abeceda TS, který rozpoznává L v polynomiálním čase a $\Sigma \subseteq \Gamma \setminus \varepsilon$. Zvolme libovolné kódování $k_1 : \Gamma \to \{0,1\}^{d_1}$. Pak existuje posloupnost Booleovských obvodů $\{C_n\}_{n=0}^\infty$ v bázi $\{\wedge, \vee, \neg, 0, 1\}$, která splňuje:*

1. *Obvod C_n definuje funkci nd Booleovských proměnných.*
2. *Pro každé $n \in \mathrm{N}$ a každé slovo $w \in \Sigma^*$, $|w| \leq n$ platí $w \in L \iff C_n(\mathrm{kod}_n(k_1, w)) = 1$.*
3. *Pokud $x \in \{0,1\}^{nd}$ není tvaru $\mathrm{kod}_n(k_1, w)$ pro žádné slovo $w \in \Sigma^*$, $|w| \leq n$, pak $C_n(x) = 0$.*
4. *Velikost obvodu C_n je omezena polynomem od n.*
5. *Obvod C_n lze zkonstruovat algoritmem v čase polynomiálním od n.*

Důkaz: Z předpokladů plyne, že existuje TS M, který rozpoznává L v čase, který je omezen nějakým polynomem $p(n)$. Protože vícepáskový stroj lze simulovat jednopáskovým strojem v polynomiálním čase, můžeme předpokládat, že M je jednopáskový. Množinu jeho stavů označme Q. Budeme předpokládat, že stroj před ukončením výpočtu přesune hlavu na nejlevější pozici na pásce. Kromě toho budeme předpokládat, že po dosažení koncového stavu výpočet formálně pokračuje dále, ale konfigurace se nemění.

Zvolme nějakou délku vstupu n a označme $\ell = p(n)$, což je omezení počtu kroků výpočtu i počtu polí pásky, které výpočet může využít. Výpočet stroje

M pro konkrétní vstupní slovo w budeme reprezentovat tabulkou obsahující posloupnost konfigurací. Každá konfigurace bude tvořena dvěma řádky délky ℓ, které odpovídají slovům $x_1 \dots x_\ell$ a $y_1 \dots y_\ell$ z Definice 2.5.2. Znaky těchto slov budou kódovány pomocí vektorů bitů. Tabulka bude obsahovat ℓ dvojic řádků reprezentující jednotlivé konfigurace. Znaky na řádcích, které obsahují kopie pracovní pásky, budou kódovány pomocí kódování k_1 z předpokladů věty. Znaky na řádcích obsahujících stavová slova jednotlivých konfigurací budou kódována pomocí nějakého kódování $k_2 : Q \cup \{\varepsilon\} \to \{0,1\}^{d_2}$ pro vhodné d_2. Podle předpokladu dojde výpočet M do koncového stavu nejpozději po ℓ krocích a pak se konfigurace nemění. Poslední řádek tabulky tedy obsahuje konfiguraci s koncovým stavem.

Protože M je deterministický, lze každou konfiguraci v tabulce kromě první odvodit z konfigurace předchozí. Tvrdíme, že odvození aktuální konfigurace z předchozí konfigurace lze provést lokálně, což znamená, že symboly x_i a y_i v aktuální konfiguraci lze odvodit ze symbolů x_{i-1}, x_i, x_{i+1} a y_{i-1}, y_i, y_{i+1} předchozí konfigurace. Pro jednoznačnost budeme x_i a y_i v aktuální konfiguraci zapisovat jako x'_i a y'_i.

Symbol x'_i lze určit následovně. Pokud je na i-té pozici stavového slova předchozí konfigurace zapsán stav, tedy pokud $y_i \in Q$, pak x'_i je symbol zapsaný na pásku podle přechodové tabulky stroje M na základě x_i a y_i. Pokud $y_i = \varepsilon$, pak se symbol na pásce na pozici i nemění a je tedy $x'_i = x_i$.

Symbol y'_i stavového slova aktuální konfigurace lze určit následovně. Pokud $y_{i-1} = y_i = y_{i+1} = \varepsilon$, pak $y'_i = \varepsilon$. Pokud pro některé $j \in \{i-1, i, i+1\}$ je $y_j \in Q$ a přechodová funkce M určuje přesun čtecí hlavy z pozice j na pozici i, pak y'_i je stav určený přechodovou funkcí. Pokud se čtecí hlava přesune na jinou pozici než i, je $y_i = \varepsilon$.

Popsaný výpočet x'_i a y'_i na základě x_{i-1}, x_i, x_{i+1} a y_{i-1}, y_i, y_{i+1} lze vyjádřit obvodem, jehož vstupem jsou kódy $k_1(x_{i-1})$, $k_1(x_i)$, $k_1(x_{i+1})$, $k_2(y_{i-1})$, $k_2(y_i)$, $k_2(y_{i+1})$ a výstupem kódy $k_1(x'_i)$ a $k_2(y'_i)$. Tento obvod má $3d_1 + 3d_2$ vstupních bitů a $d_1 + d_2$ výstupních bitů. Podle Věty 2.2.4 lze požadovanou funkci vyjádřit takovýmto obvodem velikosti nejvýše $(d_1 + d_2)2^{3d_1+3d_2+2}$. Pro účely dokazované věty je tento odhad konstanta, protože závisí pouze na TS M a nezávisí na délce vstupního slova.

Pro vstupní slovo $w \in \Sigma^*$, $|w| \leq n$ je první konfigurace tvořena obsahem pracovní pásky $w\varepsilon^{\ell-|w|}$ a stavovým slovem $q_0\varepsilon^{\ell-1}$, které určuje, že stav řídící jednotky je počáteční a čtecí hlava je umístěna na první pozici. Prvních n znaků pracovní pásky závisí na slově w. Pokud je w kratší než n znaků, vzniknou tyto znaky doplněním w na délku n prázdnými symboly. Kód prvních n znaků pracovní pásky tedy je $\mathrm{kod}_n(k_1, w)$. Zbývajících $\ell - n$ znaků pracovní pásky a znaky stavového slova na w nezávisí. Z hlediska obvodu, který konstruujeme, jsou tedy kódy těchto znaků konstanty.

Popíšeme ještě postup výpočtu kódů dalších konfigurací. Slova popisující jednu konfiguraci se skládají z ℓ pozic, přičemž pozice i je popsána znaky x_i a y_i. Kódy těchto znaků pro $2 \leq i \leq \ell - 1$ budou počítat kopie výše popsaného obvodu, jejichž vstupem jsou bity popisující pozice $i-1, i, i+1$ předchozí konfigurace. Pro $i = 1$ neobsahuje předchozí konfigurace pozici $i - 1$ a pro $i = \ell$ neobsahuje předchozí konfigurace pozici $i + 1$. V těchto případech bude pozice i

počítána obvodem, ve kterém jsou vstupní bity chybějících pozic nahrazeny libovolnými konstantami a které tedy závisí pouze na $2d_1 + 2d_2$ vstupních bitech. V tabulce je $\ell - 1$ konfigurací, které jsou počítány tímto způsobem. Obvod pro výpočet těchto konfigurací tedy obsahuje $(\ell-1)\ell$ kopií výše popsaného obvodu, z nichž každá má velikosti nejvýše $(d_1+d_2)2^{3d_1+3d_2+2}$. Celková velikost obvodů počítajících tabulku konfigurací je tedy $O(\ell^2)$.

Jako další krok je třeba doplnit obvod, jehož vstupem jsou bity první pozice ve stavovém slově poslední konfigurace a který vypočte výstupní bit hlavní části obvodu podle stavu řídící jednotky, který je určen poslední konfigurací. Pokud je tento stav přijímající, je výstup 1, pokud je zamítající, je výstup 0. Tím vznikne obvod, který označíme $C'_n(x)$, jehož vstupní vektor bitů $x \in \{0,1\}^{nd}$ je tvořen kódem prvních n pozic obsahu pracovní pásky v první konfiguraci. Pokud je do vstupu obvodu dosazen $\mathrm{kod}_n(k_1, w)$, pak výstup obvodu je 1 právě tehdy, když $w \in L$. Velikost obvodu C'_n je $O(\ell^2) = O(p^2(n))$.

Požadovaný obvod C_n vznikne jako konjunkce obvodu C'_n a obvodu, který testuje, že vstupní vektor x je tvaru $\mathrm{kod}_n(k_1, w)$ pro nějaké $w \in \Sigma^*$, $|w| \leq n$. Tento test lze vyjádřit jako konjunkci následujících podmínek. Vstupní vektor x je tvořen n bloky d bitů. První podmínka je, že každý z těchto bloků kóduje znak z množiny $\Sigma \cup \{\varepsilon\}$. Pro jeden blok lze tuto podmínku ověřit obvodem, jehož velikost závisí pouze na d, tedy pro všechny bloky obvodem velikosti $O(n)$. Kromě toho je potřeba pro každé $i = 1, \ldots, n-1$ ověřit implikaci, že pokud i-tý blok kóduje ε, pak také $i+1$-ní blok kóduje ε. Pro každé i lze tuto podmínku ověřit obvodem, jehož velikost závisí pouze na d, a pro celý vstup x tedy obvodem velikosti $O(n)$. Ověření konjunkce uvedených podmínek je tedy také vyjádřitelné obvodem velikosti $O(n)$.

Velikost obvodu C_n je $O(p^2(n)) + O(n)$, tedy je omezena polynomem od n. Protože popsanou konstrukci lze provést v čase polynomiálním od n, je věta dokázána. □

3 Třída NP

Třída NP je třídou úloh, které mají následující vlastnost. Pokud má libovolná instance úlohy kladnou odpověď, pak tento fakt lze doložit "důkazem", který lze zapsat jako posloupnost znaků nejvýše polynomiální délky vůči délce instance, a navíc, jeho správnost lze ověřit v polynomiálním čase. Smysl pojmu "důkaz" v tomto případě vysvětlíme na příkladech.

Uvažme nejprve již dříve zmíněný problém kliky v grafu.

Název: Problém kliky.
Vstup: graf $G = (V, E)$, číslo k.
Výstup: Existuje v G klika velikosti aspoň k?

Má-li nějaká instance $\langle G, k\rangle$ této úlohy kladnou odpověď, je tento fakt možné doložit tím, že předložíme množinu k vrcholů, která tvoří kliku. Tato množina pak slouží jako důkaz toho, že uvažovaná instance má kladnou odpověď. K ověření toho, že je důkaz správný, stačí ověřit, že předložená množina obsahuje

aspoň k prvků a je skutečně klikou. K tomu stačí projít všechny dvojice vrcholů z množiny a ověřit, že jsou spojeny hranou, což lze provést v čase nejvýše kvadratickém od počtu vrcholů grafu.

Jiný příklad je rozpoznávání Booleovských formulí v CNF, které jsou splnitelné, tj. jejich negace není tautologie. Tento problém se označuje zkratkou SAT, která odpovídá anglickému slovu satisfiability, tj. splnitelnost.

Název: Problém SAT.
Vstup: Booleovská formule v CNF.
Výstup: Je daná formule splnitelná?

Zjištění, zda daná formule je splnitelná vyžaduje v obecném případě exponenciálně mnoho kroků. Pokud už ale máme nalezeno ohodnocení proměnných, pro které je formule splněna, můžeme se o správnosti ohodnocení snadno přesvědčit i bez znalosti postupu, kterým bylo ohodnocení proměnných nalezeno. Ověření toho, že dané ohodnocení formuli splňuje, lze provést v čase lineárním v délce formule.

V případě problému kliky bude důkazem libovolná klika velikosti alespoň k. V případě problému SAT je důkazem libovolné ohodnocení proměnných, které splňuje danou formuli. Uvedené úlohy jsou typickými reprezentatny třídy NP. Třída NP zobecňuje uvedené příklady v tom, že jako "důkaz" povolíme libovolnou posloupnost znaků a jako verifikační proceduru povolíme libovolný polynomiální algoritmus.

3.1 Zavedení NP pomocí existenční kvantifikace

Protože budeme pracovat s relacemi, je potřeba zvolit nějaké kódování pro dvojice slov. Pro jednoduchost budeme libovolnou dvojici slov $\langle x, y\rangle$, kde $x \in \Sigma_1^*$ a $y \in \Sigma_2^*$ kódovat jako $x\#y$, kde předpokládáme, že $\# \notin \Sigma_1 \cup \Sigma_2$.

Jazyk, jehož prvky jsou dvojice slov, budeme nazývat relací. Relaci R nazveme polynomiálně omezenou, jestliže existuje polynom $p(n)$ tak, že kdykoli $x\#y \in R$, pak $|y| \leq p(|x|)$. Místo polynomiálně omezená budeme pro stručnost psát p-omezená.

Definice 3.1.1 Jazyk L patří do NP právě tehdy, když existuje p-omezená relace $R \in \mathrm{P}$ tak, že pro každé slovo w platí $w \in L \iff (\exists x)\ w\#x \in R$.

Chceme-li podle této definice dokázat, že problém kliky patří do NP, je třeba zvolit kódování instancí, tj. dvojic $\langle G, k\rangle$, a množin vrcholů. Jako R pak vezmeme množinu všech dvojic $w\#v$, kde w je kód instance, v je kód množiny vrcholů a navíc, množina kódovaná slovem v je klikou, která dává kladnou odpověď na instanci kódovanou slovem w. Lze snadno ověřit, že potřebná kódování lze zvolit tak, že získaná relace R splňuje všechny podmínky definice 3.1.1. Z toho plyne, že množina všech instancí problému kliky, které mají kladnou odpověď, patří do NP.

Analogicky, problém SAT, tj. problém splnitelnosti Booleovských formulí v CNF, je v NP. K důkazu je třeba zvolit vhodné kódování formulí a vhodné kódování ohodnocení proměnných. Pak utvoříme relaci R, která bude obsahovat

právě všechny dvojice $w\#v$, kde w je kód formule a v je splňující ohodnocení jejích proměnných. Protože potřebná kódování lze zvolit tak, aby R splňovala podmínky Definice 3.1.1, kde jako L bereme množinu všech splnitelných Booleovských formulí v CNF, patří problém SAT do NP.

Pro pozdější použití zaveďme ještě následující třídu.

Definice 3.1.2 *Libovolný jazyk L nad libovolnou abecedou Σ patří do coNP právě tehdy, když $\Sigma^* \setminus L$ patří do NP.*

3.2 Polynomiální převoditelnost

Definice 3.2.1 Řekneme, že jazyk L_1 v abecedě Σ_1 je polynomiálně převoditelný na L_2 v abecedě Σ_2, jestliže existuje funkce $f : \Sigma_1^* \to \Sigma_2^*$ vyčíslitelná v polynomiálním čase a taková, že pro každé slovo w v abecedě Σ_1 platí

$$w \in L_1 \iff f(w) \in L_2.$$

Místo polynomiální převoditelnost budeme pro stručnost psát p-převoditelnost. Vztah p-převoditelnosti umožňuje přenášet polynomiální algoritmy z jedné úlohy na jinou úlohu. Přesná formulace je v následující větě.

Věta 3.2.2 *Je-li L_1 p-převoditelný na L_2 a $L_2 \in \mathrm{P}$, pak je také $L_1 \in \mathrm{P}$.*

Důkaz: Nechť $f(w)$ je vypočitatelná v čase $p_f(|w|)$, kde p_f je polynom. Nechť pro libovolné v lze rozhodnutí, zda $v \in L_2$ zjistit v čase $p_2(|v|)$, kde p_2 je polynom. Popišme polynomiální algoritmus pro rozpoznávání L_1. Pro vstup w nejprve vypočítejme $f(w)$ a pak rozhodněme, zda $f(w) \in L_2$. Protože délka $f(w)$ je nejvýše $p_f(|w|)$, je celkový čas potřebný k tomuto výpočtu nejvýše $p_f(|w|) + p_2(p_f(|w|))$. Tím je věta dokázána, protože tento čas je polynomiální. □

Vztah p-převoditelnosti je tranzitivní.

Věta 3.2.3 *Jestliže L_1 je p-převoditelný na L_2 a L_2 je p-převoditelný na L_3, pak L_1 je p-převoditelný na L_3.*

Důkaz: Nechť p-převoditelnost L_1 na L_2 je realizována funkcí f_1 a p-převoditelnost L_2 na L_3 je realizována funkcí f_2. Požadovanou p-převoditelnost L_1 na L_3 pak realizuje složení funkcí $f_2(f_1(w))$, protože

$$w \in L_1 \iff f_1(w) \in L_2 \iff f_2(f_1(w)) \in L_3.$$

Ještě ověříme, že uvedená složená funkce je vyčíslitelná v polynomiálním čase. Je-li $t_1(n)$ čas výpočtu f_1 a $t_2(n)$ čas výpočtu f_2, pak $t_1(n) + t_2(t_1(n))$ je horní odhad času výpočtu $f_2(f_1(w))$. Tento odhad je opět polynom. □

Uveďme si jednoduché příklady p-převoditelnosti.

Věta 3.2.4 *Problém SAT je p-převoditelný na problém omezené splnitelnosti monotonní Booleovské formule.*

Důkaz: Nechť φ je libovolná Booleovská formule v CNF s proměnnými $x_1, \ldots, x_n$. Vytvořme monotonní formuli φ' s proměnými $x_1, \ldots, x_n$ a $y_1, \ldots, y_n$, která vznikne z φ nahrazením každého výskytu $\neg x_i$ za y_i. Dále, nechť $\varphi'' = \varphi' \wedge \bigwedge_{i=1}^{n}(x_i \vee y_i)$ a definujme funkci f tak, že $f(\varphi)$ je instance problému omezené splnitelnosti monotonní Booleovské formule (φ'', n).

Pokud je φ splnitelná ohodnocením $x_i = a_i$, pak φ'' je splnitelná ohodnocením $x_i = a_i$ a $y_i = \neg a_i$, ve kterém je právě n jedniček. Tedy instance (φ'', n) je splnitelná.

Na druhé straně, předpokládejme, že instance (φ'', n) je splnitelná, tedy, že formule φ'' je splnitelná ohodnocením $x_i = a_i$ a $y_i = b_i$, které obsahuje nejvýše n jedniček. Protože pro každé i je splněna disjunkce $a_i \vee b_i$, je pro každé i právě jedna z hodnot a_i a b_i rovna 1, tedy platí $b_i = \neg a_i$. Z toho plyne, že ohodnocení $x_i = a_i$ splňuje formuli φ. □

Věta 3.2.5 *Problém SAT je p-převoditelný na problém kliky.*

Důkaz: Nechť $\varphi = c_1 \wedge \ldots \wedge c_m$ je libovolná Booleovská formule v CNF, kde c_i pro $i = 1, \ldots, m$ jsou její klausule. Napišme klausule φ pod sebe, přičemž j-tý literál i-té klausule budeme značit $l_{i,j}$. Dostaneme

$c_1 = l_{1,1} \vee l_{1,2} \vee \ldots \vee l_{1,k_1}$
$c_2 = l_{2,1} \vee l_{2,2} \vee \ldots \vee l_{2,k_2}$
.
.
.
$c_m = l_{m,1} \vee l_{m,2} \vee \ldots \vee l_{m,k_m}$

Uvažme graf, jehož vrcholy jsou literály v této formuli. Dva literály jsou spojeny hranou právě tehdy, když nejsou v téže klausuli a nejsou navzájem komplementární, tj. nejsou negací jeden druhého. Tvrdíme, že v získaném grafu je klika velikosti m právě tehdy, když vstupní formule je splnitelná.

Nechť v grafu je klika velikosti m. Pak tato klika obsahuje po jednom literálu z každé klausule. Navíc, každá proměnná, která se v klice vyskytuje, se tam vyskytuje buď jen positivně nebo jen negativně. Lze tedy zvolit ohodnocení proměnných takové, že všechny literály v klice jsou splněny. Je tedy splněna každá klausule.

Nechť naopak je formule splnitelná. Pak zvolme splňující ohodnocení a v každé klausuli vyberme jeden splněný literál. Tyto literály zřejmě tvoří kliku velikosti m. □

Název: Problém vrcholového pokrytí
Vstup: Graf $G = (V, E)$, číslo k.
Výstup: Existuje množina $A \subseteq V$ nejvýše k vrcholů taková, že každá hrana G má aspoň jeden koncový vrchol v A?

Věta 3.2.6 *Problém kliky a problém vrcholového pokrytí jsou na sebe navzájem p-převoditelné.*

Důkaz: Pro libovolný graf $G = (V, E)$ označme jako $\overline{G}$ jeho komplement, tj. graf $\overline{G} = (V, \overline{E})$, v němž jsou dva vrcholy spojeny hranou právě tehdy, když nejsou spojeny hranou v G. Množinu vrcholů, z nichž žádné dva nejsou spojeny hranou, nazveme nezávislou množinou.

Množina $A \subseteq V$ je klika v G právě tehdy, když A je nezávislá v $\overline{G}$. To, že A je nezávislá v $\overline{G}$ lze ekvivalentně vyjádřit tak, že každá hrana $\overline{G}$ má aspoň jeden koncový vrchol ve $V \setminus A$. Jinak řečeno, $A \subseteq V$ je klika v G právě tehdy, když $V \setminus A$ je vrcholové pokrytí $\overline{G}$. Protože $|A| \geq k$ platí právě tehdy, když $|V \setminus A| \leq |V| - k$, máme následující polynomiální převod problému kliky na problém vrcholového pokrytí.

Definujme funkci f, která libovolné instanci $\langle G, k\rangle$ problému kliky přiřadí instanci problému vrcholového pokrytí $f(\langle G, k\rangle) = \langle \overline{G}, |V| - k\rangle$. Tvrzení z předchozího odstavce zaručuje, že v grafu $G = (V, E)$ existuje klika velikosti aspoň k právě tehdy, když v grafu $\overline{G} = (V, \overline{E})$ existuje vrcholové pokrytí velikosti nejvýše $|V| - k$. Výchozí instance problému kliky má tedy kladnou odpověď právě tehdy, když získaná instance problému vrcholového pokrytí má kladnou odpověď.

Funkce f je vyčíslitelná v polynomiálním čase. Tím jsme dokázali p-převoditelnost problému kliky na problém vrcholového pokrytí. Protože funkce f je inverzní sama k sobě, tj. $f(f(w)) = w$, je také problém vrcholového pokrytí p-převoditelný na problém kliky. (Pokud w nekóduje dvojici graf a číslo, dodefinujme třeba $f(w) = w$.) Tím je důkaz věty ukončen. □

3.3 NP-úplnost

Definice 3.3.1 Jazyk L nazveme NP-úplný, jestliže
1. L patří do NP;
2. každý jazyk v NP je na L polynomiálně převoditelný.

Z definice lze snadno dokázat následující tvrzení, které ukazuje důležitou vlastnost NP-úplných úloh.

Věta 3.3.2 *Nechť L je libovolná* NP*-úplná úloha. Pak $L \in$ P právě tehdy, když* P = NP.

Důkaz: Pokud $L \in$P, pak z druhé podmínky na NP-úplnost a z Věty 3.2.2 plyne, že každá úloha z NP je v P.

Pokud P=NP, pak $L \in$ P, protože $L \in$NP. □

Ukážeme si příklady několika NP-úplných úloh. Uvědomme si, že kdyby některá NP-úplná úloha měla polynomiální algoritmus, pak všechny úlohy v NP by měly polynomiální algoritmus. Jinak řečeno, bylo by P=NP. Toto se považuje za nepravděpodobné. Proto se důkaz NP-úplnosti nějaké úlohy považuje za argument ve prospěch toho, že úloha nemá polynomiální algoritmus.

Problémem CSAT (circuit SAT) budeme rozumět úlohu pro daný Booleovský obvod $C(x_1, \ldots, x_n)$ rozhodnout, zda existuje ohodnocení vstupních proměnných, pro které je výstup obvodu 1.

Věta 3.3.3 *Problém CSAT je* NP*-úplný.*

Důkaz: Nechť L je libovolný jazyk z NP. Označme jako R některou relaci podle Definice 3.1.1 pro jazyk L a nechť $p(n)$ je polynom, pro který je R p-omezená. Nechť $\{C_n\}_{n=0}^{\infty}$ je posloupnost obvodů zaručená Větou 2.5.3 pro jazyk R při nějakém kódování $k : \Gamma \to \{0,1\}^d$, kde Γ zahrnuje abecedu pro relaci R. Pro dané slovo w označme $m = p(|w|)$ a označme jako C_w obvod $C_{|w|+1+m}$, ve kterém jsou do prvních $d(|w|+1)$ vstupů dosazeny konstanty odpovídající kódu slova $w\#$ a zbylé vstupy jsou volné. Obvod C_w lze k danému slovu w zkonstruovat v polynomiálním čase. Z omezenosti relace R polynomem $p(|w|)$ plyne $w\#v \in R \Rightarrow |v| \leq m$. Obvod C_w má dm vstupů. Z vlastností obvodu $C_{|w|+1+m}$, které zaručuje Věta 2.5.3, plyne, že pro libovolné $x \in \{0,1\}^{dm}$ platí

$$C_w(x) = 1 \iff (\exists v)\,(\mathrm{kod}_m(k, v) = x \wedge w\#v \in R)\ . \tag{6}$$

Definujme funkci f tak, že $f(w) = C_w$. Z (6) plyne, že funkce f převádí problém $w \in L$ na problém CSAT smyslu p-převoditelnosti. Tím je tvrzení věty dokázáno. □

Věta 3.3.4 *Problém SAT je* NP*-úplný.*

Důkaz: Dokážeme, že problém CSAT je p-převoditelný na problém SAT. Uvažme libovolný obvod C s proměnnými $x_1, x_2, \ldots, x_n$ v bázi $\{\wedge, \vee, \neg\}$ a označme jako m velikost C. Výpočetním uzlům obvodu (uzlům, které odpovídají spojkám) přiřadíme nové proměnné $y_1, y_2, \ldots, y_m$. Tyto proměnné budou reprezentovat hodnoty jednotlivých výpočetních uzlů. Přitom proměnnou y_m přiřadíme výstupnímu uzlu.

Zkonstruujeme formuli $\varphi_C(x_1, \ldots, x_n, y_1, \ldots, y_{m-1})$ v CNF, která bude pravdivá právě tehdy, když proměnné $y_1, \ldots, y_{m-1}$ jsou rovny hodnotám, které mají jednotlivé nevýstupní uzly C při výpočtu pro vstup $x_1, \ldots, x_n$, který dává na výstupu hodnotu 1. Existence ohodnocení $x_1, \ldots, x_n$ pro které je $C(x_1, \ldots, x_n) = 1$ je pak ekvivalentní existenci ohodnocení proměnných $x_1, \ldots, x_n, y_1, \ldots, y_{m-1}$ tak, že $\varphi_C(x_1, \ldots, x_n, y_1, \ldots, y_{m-1}) = 1$.

To, že φ_C lze zkonstruovat v CNF plyne z toho, že ověření souhlasu hodnot proměnných $y_1, \ldots, y_{m-1}$ se skutečným výpočtem obvodu lze vyjádřit jako konjunkci mnoha "lokálních" podmínek. Nejprve budeme pracovat se všemi proměnnými $y_1, y_2, \ldots, y_m$. Pro každý uzel obvodu vyjádříme podmínku, která vyjadřuje, že hodnota jemu příslušné proměnné souhlasí s hodnotami vstupů uzlu a spojkou, kterou uzel počítá.

Nechť uzel přiřazený proměnné y_r odpovídá konjunkci a nechť jeho vstupy jsou uzly přiřazené proměnným y_s a y_t. V tom případě potřebujeme vyjádřit podmínku $y_r \equiv y_s \wedge y_t$. Označme tuto podmínku jako ψ_r. Protože výsledná formule má být v CNF, vyjádříme ψ_r nejprve jako

$$\psi_r \equiv (y_r \to y_s \wedge y_t) \wedge (y_s \wedge y_t \to y_r),$$

pak ekvivalentně jako

$$\psi_r \equiv (y_r \to y_s) \wedge (y_r \to y_t) \wedge (y_s \wedge y_t \to y_r)$$

a nakonec jako

$$\psi_r \equiv (\neg y_r \vee y_s) \wedge (\neg y_r \vee y_t) \wedge (\neg y_s \vee \neg y_t \vee y_r).$$

Podobně vyjádříme formuli ψ_r v případě, že y_r odpovídá disjunkci. V tom případě dostaneme postupně

$$\psi_r \equiv (y_r \to y_s \vee y_t) \wedge (y_s \vee y_t \to y_r),$$

$$\psi_r \equiv (y_r \to y_s \vee y_t) \wedge (y_s \to y_r) \wedge (y_t \to y_r),$$

$$\psi_r \equiv (\neg y_r \vee y_s \vee y_t) \wedge (\neg y_s \vee y_r) \wedge (\neg y_t \vee y_r).$$

Pokud y_r odpovídá negaci uzlu y_s, pak dostaneme formuli

$$\psi_r \equiv (y_r \vee y_s) \wedge (\neg y_r \vee \neg y_s).$$

Dosud zkonstruované formule nezávisely na vstupech obvodu. Pokud uzly podřízené uzlu y_r jsou vstupy, liší se konstrukce v tom, že místo proměnných y_s a y_t použijeme příslušné proměnné $x_1, \ldots, x_n$.

Nechť $\varphi_C = \psi'_1 \wedge \ldots \psi'_m$, kde $\psi'_1, \ldots, \psi'_m$ vzniknou z $\psi_1, \ldots, \psi_m$ dosazením konstanty 1 za y_m. Tvrdíme, že

$$C(x_1, \ldots, x_n) \equiv (\exists y_1) \ldots (\exists y_{m-1})\, \varphi_C(x_1, \ldots, x_n, y_1, \ldots, y_{m-1}). \qquad (7)$$

Zvolme nějaké ohodnocení proměnných $x_1, \ldots, x_n$. Jestliže je pro toto ohodnocení $C(x_1, \ldots, x_n) = 1$, pak každé z proměnných $y_1, \ldots, y_{m-1}$ přiřaďme hodnotu jí odpovídajícího uzlu C. Tím dostaneme ohodnocení proměnných $y_1, \ldots, y_{m-1}$, které spolu s $y_m = 1$ splňuje všechny lokální podmínky a je tedy $\varphi_C = 1$.

Předpokládejme naopak, že pro zvolené ohodnocení $x_1, x_2, \ldots, x_n$ je pravdivá formule $(\exists y_1) \ldots (\exists y_{m-1}) \psi'_1 \wedge \ldots \wedge \psi'_m$. Vezměme některé ohodnocení proměnných $y_1, \ldots, y_m$, pro které je vnitřek formule splněn a ve kterém je $y_m = 1$. Speciálně, jsou splněny všechny lokální podmínky. Ze splnění lokálních podmínek lze dokázat indukcí od vstupů až k výstupu, že hodnota každé proměnné y_r je rovna hodnotě příslušného uzlu. Protože $y_m = 1$ je hodnota celého obvodu C, dokázali jsme, že C dává pro zvolené ohodnocení $x_1, \ldots, x_n$ výstup 1. Tím je požadovaná ekvivalence dokázána.

Z ekvivalence (7) plyne, že existence splňujícího ohodnocení pro obvod C je ekvivalentní existenci splňujícího ohodnocení pro formuli φ_C. Poznamenejme, že z toho neplyne, že C a φ_C jsou ekvivalentní.

Protože konstrukci formule φ_C lze provést v polynomiálním čase, dává tato konstrukce p-převoditelnost problému CSAT na problém SAT. Protože CSAT je NP-úplný, je také problém SAT NP-úplný. □

Důsledek 3.3.5 *Problém SAT je v P právě tehdy, když* P = NP.

Zkratka 3-SAT označuje problém SAT omezený na formule, jejichž všechny klausule mají právě tři literály.

Věta 3.3.6 *Problém SAT je p-převoditelný na 3-SAT.*

Důkaz: Nechť $c_1, c_2, \ldots, c_m$ jsou klausule libovolné instance problému SAT na množině proměnných U. Každou klausuli nahradíme skupinou klausulí ze tří literálů, která může obsahovat nové proměnné. Pro klausuli c_j přidáme množinu proměnných U_j, které se nebudou vyskytovat v žádné jiné klausuli.

Nechť c_j je složena z jediného literálu a. Pak nové klausule budou $(a \vee y_1 \vee y_2), (a \vee \neg y_1 \vee y_2), (a \vee y_1 \vee \neg y_2), (a \vee \neg y_1 \vee \neg y_2)$ a $U_j = \{y_1, y_2\}$.

Každé splňující ohodnocení původní formule splňuje literál a a tedy také všechny nové klausule. Také naopak, každé ohodnocení, které splňuje nové klausule, musí splnit i literál a. To proto, že pro každou kombinaci hodnot nových proměnných y_1, y_2 je ve čtveřici nových klausulí taková, v níž jsou oba literály z nových proměnných nesplněny.

Nechť c_j je tvořena dvěma literály a_1, a_2. Pak nové klausule budou $(a_1 \vee a_2 \vee y), (a_1 \vee a_2 \vee \neg y)$ a $U_j = \{y\}$. Analogicky jako v předchozím případě lze každé splňující ohodnocení původní klausule rozšířit na splňující ohodnocení nových klausulí. Také naopak, jsou-li nové klausule splněny nějakým ohodnocením, pak je splněna také klausule $a_1 \vee a_2$.

Je-li c_j tvořena třemi literály, necháme ji beze změny.

Pokud je c_j je tvořena k literály $a_1, \ldots, a_k$, kde $k \geq 4$, pak nové klausule budou

$$(a_1 \vee a_2 \vee y_1), (\neg y_1 \vee a_3 \vee y_2), \ldots, (\neg y_{k-4} \vee a_{k-2} \vee y_{k-3}), (\neg y_{k-3} \vee a_{k-1} \vee a_k)$$

a $U_j = \{y_1, \ldots, y_{k-3}\}$. Pokud jsou nové klausule splnitelné, pak je splnitelná i původní klausule, protože ji lze z nových klausulí odvodit pomocí rezoluce. Předpokládejme nyní, že je původní klausule splněna nějakým ohodnocením a doplňme ho tak, aby splnilo i nové klausule. Vyberme některý ze splněných literálů a označme jej a_r. Tento literál je obsažen v jedné z nových klausulí, která je tedy také splněna. Ostatní nové klausule splníme tak, že klausule obsahující literály a_s s indexem $2 \leq s < r$, splníme ohodnocením $y_{s-1} = 1$ a klausule obsahující literály a_s s indexem $r < s \leq k-1$, splníme ohodnocením $y_{s-2} = 0$. □

Důsledek 3.3.7 *Problém 3-SAT je* NP*-úplný.*

Důkaz: Tvrzení je důsledkem Vět 3.3.4 a 3.3.6. □

Poznamenejme, že v důkazu Věty 3.3.4 je zkonstruována formule, jejíž klausule mají délku nejvýše 3. K převedení otázky splnitelnosti této formule na 3-SAT tedy není potřeba 3.3.6 v plné obecnosti, ale stačí případy pro klausule délky nejvýše 3.

Již dříve jsme dokázali, že problém SAT je p-převoditelný na problém kliky a že problém kliky je p-převoditelný na problém vrcholového pokrytí. Z tranzitivity p-převoditelnosti a z toho, že SAT je NP-úplný plyne též NP-úplnost problému kliky a vrcholového pokrytí.

3.4 Další příklady NP-úplných úloh

3.4.1 Problém přesného 3-pokrytí

Název: Problém přesného 3-pokrytí
Vstup: Konečná množina X, jejíž velikost je dělitená 3, a systém M některých jejích tříprvkových podmnožin.
Výstup: Existuje podsystém $M' \subseteq M$ tak, že množiny v M' jsou disjunktní a jejich sjednocení je X, jinak řečeno, že každý prvek X je obsažen právě v jedné trojici M'?

Věta 3.4.1 *Problém přesného 3-pokrytí je* NP*-úplný.*

Důkaz: Úloha je v NP, protože ověřit splnění podmínky pro libovolnou uhodnutou podmnožinu M' lze v polynomiálním čase.

Dále dokážeme, že 3-SAT je p-převoditelný na problém přesného 3-pokrytí. Uvažme libovolnou instanci 3-SAT v podobě formule na n proměnných $x_1, x_2, \ldots, x_n$ obsahující klausule $c_1, c_2, \ldots, c_m$. Zkonstruujeme instanci přesného 3-pokrytí, která bude mít řešení právě tehdy, když uvažovaná formule je splnitelná. Množina X se bude skládat z prvků popsaných v následující tabulce

$u_{i,j}$, $\bar{u}_{i,j}$	pro $i = 1, 2, \ldots, n$ a $j = 1, 2, \ldots, m$
$a_{i,j}$, $b_{i,j}$	pro $i = 1, 2, \ldots, n$ a $j = 1, 2, \ldots, m$
r_j, s_j	pro $j = 1, 2, \ldots, m$
e_k, f_k	pro $k = 1, 2, \ldots, mn - m$

Množinu M zkonstruujeme po částech.

První část P nazvěme "ohodnocení proměnných". Tato část se bude skládat z podskupin odpovídajících jednotlivým proměnným. Pro proměnnou x_i budeme mít skupiny trojic T_i a F_i. Skupinu T_i tvoří trojice

$$\{\bar{u}_{i,j}, a_{i,j}, b_{i,j}\} \text{ pro } 1 \leq j \leq m.$$

Skupinu F_i tvoří trojice

$$\{u_{i,j}, a_{i,j-1}, b_{i,j}\} \text{ pro } 1 \leq j \leq m.$$

Odečítání jedničky v indexu v $a_{i,j-1}$ bereme cyklicky, tj. 0 považujeme za rovnou m.

Prvky $a_{i,j}$ a $b_{i,j}$, $1 \leq j \leq m$ budou náležet pouze do trojic skupin T_i a F_i. S pomocí toho ověříme, že libovolné přesné 3-pokrytí musí obsahovat právě m trojic z $T_i \cup F_i$, konkrétně buď všechny trojice z T_i nebo všechny trojice z F_i.

Lemma 3.4.2 *Jestliže M vznikne z P přidáním trojic, které neobsahují prvky $a_{i,j}, b_{i,j}$, pak libovolné přesné 3-pokrytí M' vybrané z M obsahuje pro každé i buď celou F_i a žádnou trojici z T_i nebo naopak celou T_i a žádnout trojici z F_i.*

Důkaz: Uvažme nějaké přesné pokrytí množiny X trojicemi vybranými z M a zvolme libovolné $i = 1, 2, \ldots, n$. Prvek $a_{i,1}$ může být pokryt buď trojicí $\{\bar{u}_{i,1}, a_{i,1}, b_{i,1}\}$ nebo trojicí $\{u_{i,2}, a_{i,1}, b_{i,2}\}$. Uvažme nejprve druhou možnost. Pak trojice $\{\bar{u}_{i,2}, a_{i,2}, b_{i,2}\}$ nemůže být použita, a proto prvek $a_{i,2}$ musí být

pokryt trojicí $\{u_{i,3}, a_{i,2}, b_{i,3}\}$. Pak ovšem $\{\bar{u}_{i,3}, a_{i,3}, b_{i,3}\}$ nemůže být použita a prvek $a_{i,3}$ musí být pokryt trojicí $\{u_{i,4}, a_{i,3}, b_{i,4}\}$. Takto postupně dostaneme, že jsou použity všechny trojice F_i a žádná trojice T_i. Kdybychom na začátku předpokládali, že k pokrytí $a_{i,1}$ je použita trojice $\{\bar{u}_{i,1}, a_{i,1}, b_{i,1}\}$, dostali bychom analogickým postupem, že jsou použity všechny trojice T_i a žádná trojice F_i. □

Všimněme si, že je-li z $T_i \cup F_i$ použita skupina T_i, zůstávají touto skupinou nepokryté prvky $u_{i,1}, u_{i,2}, \ldots, u_{i,m}$. Je-li použita skupina F_i, zůstávají touto skupinou nepokryté prvky $\bar{u}_{i,1}, \bar{u}_{i,2}, \ldots, \bar{u}_{i,m}$.

Dokázaný fakt využijeme k tomu, že z libovolného přesného 3-pokrytí vybraného z M odvodíme určité ohodnocení proměnných x_i. V tomto ohodnocení budou pravdivé ty literály, které nejsou pokryty. Ohodnocení určené $M' \subseteq M$ bude takové, že proměnné x_i přiřadí hodnotu true, jestliže M' obsahuje T_i a hodnotu false, jestliže M' obsahuje F_i. Další části M budou zvoleny tak, že o tomto ohodnocení bude možné dokázat, že splňuje výchozí formuli.

Druhou část M nazvěme "ověření splnitelnosti" a označme K. Skládá se z podskupin, které odpovídají jednotlivým klausulím c_j. Klausuli c_j odpovídají trojice

$$\{u_{i,j}, r_j, s_j\}, \text{ jestliže } x_i \text{ je literálem } c_j$$

a

$$\{\bar{u}_{i,j}, r_j, s_j\}, \text{ jestliže } \neg x_i \text{ je literálem } c_j.$$

Protože každá klausule obsahuje tři literály, obsahuje skupina K pro každé j právě tři trojice obsahující prvky r_j, s_j. Prvky r_j, s_j nebudou pokryty žádnou trojicí mimo skupinu K. Přestože M ještě není celá zkonstruována, můžeme na základě již popsaných částí a za uvedeného předpokladu dokázat, že existence přesného 3-pokrytí M' vybraného z M implikuje existenci splňujícího ohodnocení.

Lemma 3.4.3 *Jestliže M vznikne z $P \cup K$ přidáním trojic neobsahujících $a_{i,j}, b_{i,j}, r_j, s_j$ a lze vybrat přesné 3-pokrytí M', pak je formule $c_1 \wedge \ldots \wedge c_m$ splnitelná.*

Důkaz: Libovolné přesné 3-pokrytí $M' \subseteq M$ musí obsahovat pro každé j právě jednu trojici z K pokrývající prvky r_j, s_j. Podle uvedeného předpokladu k tomu lze použít pouze trojice ze skupiny K. Pokrytí M' tedy obsahuje jednu z nich. Tato trojice obsahuje prvek $u_{i,j}$ nebo $\bar{u}_{i,j}$ pro některé i. Tento prvek odpovídá podle konstrukce K některému literálu klausule c_j. Protože tento prvek je pokryt trojicí z K, nemůže být pokryt žádnou trojicí z $T_i \cup F_i$. Z toho plyne, že jestliže je to prvek $u_{i,j}$, pak M' nemůže obsahovat F_i. Obsahuje tedy skupinu T_i, což znamená, že výše zkonstruované ohodnocení přiřazuje $x_i = \text{true}$. Protože klausule c_j obsahuje literál x_i, je výše popsaným ohodnocením splněna. Analogickou úvahou dostaneme, že klausule c_j je splněna, jestliže trojice z K, která pokrývá r_j, s_j, je trojice $\{\bar{u}_{i,j}, r_j, s_j\}$. V tomto případě klausule c_j obsahuje $\neg x_i$, M' obsahuje F_i a $x_i = \text{false}$. Protože tuto úvahu můžeme udělat pro každé $j = 1, 2, \ldots, m$, jsou splněny všechny klausule. □

Poslední část M nazveme "dokončení" a označíme D. Tato skupina obsahuje trojice

$$\{u_{i,j}, e_k, f_k\} \text{ pro } 1 \leq k \leq mn - m,\ 1 \leq i \leq n,\ 1 \leq j \leq m$$

a

$$\{\bar{u}_{i,j}, e_k, f_k\} \text{ pro } 1 \leq k \leq mn - m,\ 1 \leq i \leq n,\ 1 \leq j \leq m$$

Nyní můžeme shrnout, že

$$M = \bigcup_{i=1}^{n} T_i \cup \bigcup_{i=1}^{n} F_i \cup K \cup D.$$

Z Lemmatu 3.4.3 ke konstrukci T_i, F_i a K plyne, že každé přesné 3-pokrytí vybrané z M určuje splňující ohodnocení výchozích klausulí. Dokažme ještě, že existence splňujícího ohodnocení implikuje existenci přesného 3-pokrytí.

Lemma 3.4.4 *Pokud je formule $c_1 \wedge \ldots \wedge c_m$ splnitelná, pak lze vybrat přesné 3-pokrytí $M' \subseteq M$.*

Důkaz: Jestliže máme nějaké ohodnocení proměnných, které splňuje výchozí klausule, zkonstruujeme přesné 3-pokrytí takto. Nejprve pro každé $i = 1, 2, \ldots, n$ vybereme všechny trojice z T_i nebo všechny trojice z F_i podle toho, jakou hodnotu má proměnná x_i. Tím jsou pokryty všechny prvky $a_{i,j}$ a $b_{i,j}$ a celkem mn prvků z $2mn$ prkvů $u_{i,j}$ a $\bar{u}_{i,j}$. Dále pro každé $j = 1, 2, \ldots, m$ vybereme v klausuli c_j jeden ze splněných literálů. Je-li to literál x_i, je $x_i = \text{true}$. V předchozím kroku tedy byla vybrána skupina T_i a ne F_i, a tedy prvek $u_{i,j}$ je doposud vybranými trojicemi nepokrytý. Ze skupiny K tedy můžeme vybrat trojici $\{u_{i,j}, r_j, s_j\}$. Je-li z c_j vybrán literál $\neg x_i$, můžeme analogickou úvahou z K vybrat trojici $\{\bar{u}_{i,j}, r_j, s_j\}$. Tím jsou pokryty všechny prvky r_j a s_j pro $j = 1, 2, \ldots, m$ a zároveň m dalších prvků $u_{i,j}$ a $\bar{u}_{i,j}$. Zbývajících $mn - m$ z těchto prvků očíslujeme čísly $1, 2, \ldots, mn - m$. Prvek s číslem k pak pokryjeme spolu s prvky e_k a f_k příslušnou trojicí z D.

Tím jsme pokryli všechny prvky X a každý právě jednou. Požadovaná implikace je tedy dokázána. □

Množina M obsahuje

$$2mn + 3m + 2mn(mn - m)$$

trojic a byla zkonstruována explicitně na základě instance 3-SAT. Na základě toho lze snadno ověřit, že M může být zkonstruována v polynomiálním čase. □

3.4.2 Problém batohu a půlení

Název:	Problém batohu
Vstup:	Přirozené číslo n, přirozená čísla (váhy) $v_1, v_2, \ldots, v_n$ a cílová váha b.
Výstup:	Existuje $I \subseteq \{1, 2, \ldots, n\}$ tak, že $\sum_{i \in I} v_i = b$?

Věta 3.4.5 *Problém batohu je* NP*-úplný.*

Důkaz: Ukážeme, že problém přesného 3-pokrytí je p-převoditelný na problém batohu. Vezměme libovolnou instanci problému přesného 3-pokrytí určenou nějakými množinami X a M. Bez újmy na obecnosti předpokládejme, že $X = \{1, 2, \ldots, m\}$. Je třeba zvolit n, váhy $v_1, v_2, \ldots, v_n$ a cílovou váhu b tak, aby podmnožina $I \subseteq \{1, 2, \ldots, n\}$ s vlastností $\sum_{i \in I} v_i = b$ existovala právě tehdy, když výchozí instance problému přesného 3-pokrytí má řešení.

Zvolme $n = |M|$ a nechť $M = \{t_1, t_2, \ldots, t_n\}$. Váhu v_i odvodíme od trojice t_i. Váhy popíšeme ve dvojkové soustavě jako čísla, jejichž zápis je složen z m bloků délky $d = \lfloor \log n \rfloor + 1$. Počet bloků je m, protože jednotlivé bloky odpovídají prvkům X. Nechť $t_i = \{j_1, j_2, j_3\}$. Pak číslo v_i má j_1-tý, j_2-tý a j_3-tý blok tvaru $0^{d-1}1$ a ostatní bloky budou mít tvar 0^d. Délka bloku d je zvolena tak, aby při sčítání žádné množiny čísel v_i nedošlo k přenosu jedničky z žádného bloku do bloku vyššího. Zvolená délka stačí proto, že máme n čísel v_i. Sčítáme-li tedy nějakou skupinu čísel v_i, bude počet jedniček, které se sečtou v jednom bloku, nejvýše n. To je číslo, jehož zápis se vejde do $d = \lfloor \log n \rfloor + 1$ binárních číslic.

Nechť $M' \subseteq M$ je nějaký podsystém M. Nechť $I \subseteq \{1, 2, \ldots, n\}$ je množina indexů prvků z M', tj. $M' = \{t_i; i \in I\}$. Sečteme-li všechna čísla v_i pro $i \in I$, pak v j-tém bloku součtu bude součet j-tých bloků čísel v_i pro $i \in I$. Protože j-tý blok v_i je $0^{d-1}1$ pokud $j \in t_i$ a 0^d jinak, bude v j-tém bloku součtu počet výskytů j v množinách z M'. Přesněji, pokud podsystém M' obsahuje k množin, které obsahují j, bude v j-tém bloku součtu binární zápis čísla k. Speciálně, pokud jsou množiny v M' disjunktní a jejich sjednocení pokrývá celou X, má každé číslo j právě jeden výskyt v množinách z M', a tedy uvažovaný součet bude ve všech blocích obsahovat $0^{d-1}1$. Toto číslo označme b.

Tím je potřebná instance problému batoh zkonstruována. Číslo b bylo zvoleno tak, že má-li výchozí instance problému přesného 3-pokrytí řešení, pak i zkonstruovaná instance problému batohu má řešení. Dokažme opačnou implikaci.

Předpokládejme, že pro nějakou množinu I je $\sum_{i \in I} v_i = b$. Protože při sčítání nedochází k přenosům mezi bloky, jsou čísla v_i, $i \in I$ nutně taková, že platí následující. Pro každé $j = 1, 2, \ldots, m$ existuje právě jedno $i \in I$ tak, že j-tý blok v_i obsahuje $0^{d-1}1$. Jinak řečeno, právě jedna z trojic M' obsahuje j. To znamená, že výchozí instance problému přesného 3-pokrytí má řešení.

Zbývá ukázat, že všechna čísla, která jsou součástí konstruované instance problému batohu lze zkonstruovat v polynomiálním čase od velikosti výchozí instance přesného 3-pokrytí. To plyne z toho, že binární zápisy těchto čísel byly popsány explicitně na základě trojic M a z toho, že počet čísel v_i i jejich délka jsou omezeny polynomem od velikosti výchozí instance. □

Název:	Problém půlení
Vstup:	Přirozené číslo n a přirozená čísla (váhy) $v_1, v_2, \ldots, v_n$.
Výstup:	Existuje $I \subseteq \{1, 2, \ldots, n\}$ tak, že $\sum_{i \in I} v_i = \frac{1}{2} \sum_{i=1}^{n} v_i$?

Pokud řešení existuje, pak je také $\sum_{i \in I} v_i = \sum_{i \notin I} v_i$. Jde tedy o rozdělení na dvě stejně veliké části.

Věta 3.4.6 *Problém půlení je* NP*-úplný.*

Důkaz: Převodem z problému batohu. Nechť čísla $v_1, \ldots, v_n, b$ určují instanci problému batohu. Pro dostatečně veliké číslo M uvažme instanci problému půlení ve tvaru

$$\left(v_1, \ldots, v_n, M - b, M - \sum_{i=1}^{n} v_i + b\right) \tag{8}$$

Polovina součtu čísel v této instanci je M. Součet posledních dvou čísel je $2M - \sum_{i=1}^{n} v_i$. Pokud zvolíme

$$M > \sum_{i=1}^{n} v_i \ ,$$

dostaneme

$$2M - \sum_{i=1}^{n} v_i > M$$

a tedy poslední dvě čísla v (8) nemohou být současně v jedné části. Ke splnění výše uvedené nerovnosti zvolme $M = \sum_{i=1}^{n} v_i + 1$.

Pokud existuje řešení úlohy půlení (8), pak uvažme tu část, která obsahuje $M - b$. Spolu s $M - b$ musí v této části být podmnožina čísel $v_1, v_2, \ldots, v_n$, které dávají v součtu b. Existuje tedy i řešení výchozího problému batohu.

Provedením uvedené konstrukce v opačném směru dostaneme implikaci, že z existence řešení výchozího problému batohu plyne existence řešení problému půlení (8).

Protože (8) lze zkonstruovat v polynomiálním čase, dokázali jsme p-převoditelnost problému batohu na problém půlení. □

3.4.3 Problém Hamiltonovské kružnice

Název: Problém Hamiltonovské kružnice
Vstup: Graf $G = (V, E)$.
Výstup: Existuje v G uzavřená cesta, která prochází každým vrcholem právě jednou?

Věta 3.4.7 *Problém Hamiltonovské kružnice je* NP*-úplný.*

Důkaz: Dokážeme, že problém vrcholového pokrytí je p-převoditelný na problém Hamiltonovské kružnice. Nechť $\langle G = (V, E), k\rangle$ je libovolná instance problému vrcholového pokrytí. Bez újmy na obecnosti můžeme předpokládat, že graf neobsahuje vrcholy stupně 0, protože vrcholy stupně 0 lze vypustit beze změny odpovědi.

Budeme konstruovat graf $G' = f(\langle G, k\rangle)$ následovně. Pokud G obsahuje nejvýše k vrcholů nenulového stupně, bude G' libovolný pevně zvolený graf obsahující Hamiltonovskou kružnici. Protože G v tomto případě určitě obsahuje vrcholové pokrytí velikosti nejvýše k (například všechny vrcholy), splní f v tomto případě podmínku na požadovanou převoditelnost.

Pokud G obsahuje více než k uzlů nenulového stupně, zkonstruujeme graf G' s $k + 12|E|$ vrcholy následovně. V G' bude k pomocných vrcholů $v_1, \ldots, v_k$. Další vrcholy G' je výhodné považovat za body v celočíselné mříži popsané dvěma souřadnicemi. Očíslujme hrany grafu G od 0 do $|E| - 1$ a vrcholy od 0 do $|V| - 1$. Nechť e je číslo hrany a v_1, v_2 jsou čísla jejích vrcholů. Hraně e přiřadíme body $[v_1, 6e + i]$ a $[v_2, 6e + i]$ pro $i = 0, \ldots, 5$, které budou vrcholy grafu G'. Mezi těmito vrcholy vytvoříme hrany

$$([v_1, 6e + 0], [v_2, 6e + 2])$$
$$([v_1, 6e + 2], [v_2, 6e + 0])$$
$$([v_1, 6e + 3], [v_2, 6e + 5])$$
$$([v_1, 6e + 5], [v_2, 6e + 3])$$

a dále hrany $([v_j, 6e + i], [v_j, 6e + i + 1])$ pro $j = 1, 2$ a $i = 0, \ldots, 4$. Body odpovídající jedné hraně lze rozdělit na dvě skupiny po 6 bodech se stejnou první souřadnicí. Body v každé z těchto dvou skupin tvoří cestu délky 6, která je spojena čtyřmi hranami s body ve druhé skupině.

Jestliže v je vrchol stupně d v grafu G, pak z něho vychází d hran, kterým v G' odpovídá celkem $12d$ bodů. Polovina z těchto bodů, tj. $6d$, má první souřadnici v. Tyto body tvoří d cest délky 6 popsaných výše. Těchto d cest propojíme $d - 1$ dalšími hranami do jedné cesty, na které jsou body uspořádány vzestupně podle hodnoty druhé souřadnice. Tuto cestu označíme $C(v)$ a za její první (poslední) vrchol budeme považovat vrchol s nejmenší (největší) druhou souřadnicí. Rozlišení prvního a posledního bodu je jen pomocné, protože cesta je složena z neorientovaných hran.

Tímto způsobem jsme získali $|V|$ cest $C(v)$ odpovídajících jednotlivým vrcholům $v \in V$. Graf G' dokončíme tím, že první i poslední bod každé z těchto cest spojíme se všemi vrcholy $v_1, \ldots, v_k$. Celou konstrukci grafu G' lze provést v polynomiálním čase.

Dokážeme ještě, že v G' je Hamiltonovská kružnice právě tehdy, když v původním grafu G existovalo vrcholové pokrytí velikosti nejvýše k. Pokud v G existuje vrcholové pokrytí velikosti nejvýše k, pak jej doplníme na vrcholové pokrytí $U \subseteq V$ velikosti právě k. Označme vrcholy U jako $u_1, \ldots, u_k$ v libovolném pořadí.

Hamiltonovskou kružnici v G' dostaneme ve dvou krocích. V prvním kroku spojíme pro každé $i = 1, \ldots, k$ vrchol v_i s prvním uzlem cesty $C(u_i)$ a poslední uzel $C(u_i)$ spojíme s vrcholem v_{i+1} (bráno cyklicky). Tím získáme v G' uzavřenou cestu, která prochází všemi vrcholy v_i a cestami $C(u_i)$, ale neprochází cestami, které odpovídají vrcholům z $V \setminus U$. Ve druhém kroku cestu upravíme tak, aby procházela všemi body. Každý bod, kterým cesta neprochází, má tvar $[v, e + i]$, kde $v \in V \setminus U$, e je číslo některé hrany vycházející z v a $i \in \{0, \ldots, 5\}$. Nechť u je druhý vrchol hrany e. Tento vrchol patří do U a dosud zkonstruovaná cesta tedy obsahuje cestu $C(u)$, která prochází body $[u, e + i]$ pro $i = 1, \ldots, 5$. Hranu $([u, e+2], [u, e+3])$ z konstruované cesty vypustíme a nahradíme ji cestou $[u, e+2], [v, e+0], \ldots, [v, e+5], [u, e+3]$. Opakováním tohoto postupu dostaneme uzavřenou cestu, která prochází všemi vrcholy (body) G'.

Pokud v G' existuje Hamiltonovská kružnice, musí procházet všemi vrcholy v_i. Pokud tyto body a z nich vycházející hrany vypustíme, dostaneme k úseků. Z vrcholů v_i vedou hrany jen do koncových bodů cest $C(u)$ pro $u \in V$. Proto každý ze získaných úseků začíná v některém koncovém bodě některé cesty $C(u)$. Rozborem možností, jak může cesta grafem procházet, lze ověřit, že úsek, který začíná v koncovém bodě cesty $C(u)$ končí ve druhém koncovém bodě cesty $C(u)$. Protože máme celkem k úseků Hamiltonovské kružnice, lze najít k vrcholů $u_1, \ldots, u_k$ grafu G tak, že i-tý úsek začíná a končí v koncových bodech $C(u_i)$. Z tvaru grafu G' dále vyplývá, že i-tý úsek Hamiltonovské kružnice se může od $C(u_i)$ lišit jen v tom, že některé hrany $C(u_i)$ jsou vypuštěny a nahrazeny "odbočkami" právě v tom tvaru, jaký byl použit v první části důkazu věty. Úsek Hamiltonovské kružnice odvozený z cesty $C(u_i)$ tedy může procházet pouze těmi body, které mají jako první souřadnici buď u_i nebo některý vrchol, který je s u_i spojen hranou v G. Každý bod v grafu G je první souřadnicí alespoň jednoho bodu G' a tímto bodem prochází některý úsek Hamiltonovské kružnice. Vrchol v je tedy buď některým z bodů $u_1, \ldots, u_k$ nebo je s některým z těchto bodů spojen hranou v G. Z toho plyne, že vrcholy $u_1, \ldots, u_k$ tvoří vrcholové pokrytí G. □

3.4.4 Problém obchodního cestujícího

Název: Problém obchodního cestujícího
Vstup: Symetrická matice nezáporných čísel $\{d_{i,j}\}_{i,j=1}^n$ s nulami na diagonále, číslo M
Výstup: Existuje permutace $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ tak, že při dodefinování $\pi(n+1) = \pi(1)$ platí

$$\sum_{i=1}^{n} d_{\pi(i),\pi(i+1)} \leq M \,?$$

Existuje několik variant této úlohy podle toho, jaké doplňující požadavky splňuje matice čísel $d_{i,j}$:

1. obecná úloha, kdy nejsou kladeny žádné další požadavky;
2. metrická varianta, kdy požadujeme splnění trojúhelníkové nerovnosti ($d_{i,j} \leq d_{i,k} + d_{k,j}$);
3. Euklidovská varianta, kdy jsou zadány body $A_1, \ldots, A_n$ v rovině a $d_{i,j}$ je vzdálenost A_i a A_j.

Všechny tyto varianty jsou NP-těžké, tj. každá úloha z NP je na ně p-převoditelná. To, zda patří do NP (a jsou tedy NP-úplné) závisí na oboru hodnot čísel $d_{i,j}$ a na způsobu jejich zadání.

Věta 3.4.8 *Metrická varianta problému obchodního cestujícího je* NP*-těžká.*

Důkaz: Ukážeme, že problém Hamiltonovské kružnice je p-převoditelný na metrický problém obchodního cestujícího. Jestliže G je graf na n vrcholech,

pak $f(G)$ definujeme jako matici čísel

$$d_{i,j} = \begin{cases} 0 & i = j \\ 1 & i \neq j,\ (i,j) \in E \\ 2 & i \neq j,\ (i,j) \notin E \end{cases}$$

a číslo $M = n$. Lze snadno ověřit, že takto popsaná instance problému obchodního cestujícího je metrická a navíc má řešení právě tehdy, když původní graf obsahoval Hamiltonovskou kružnici. □

3.4.5 NP-úplnost MAX-2-SAT

Název: MAX-2-SAT
Vstup: Formule φ s klausulemi délky 1 a 2, přirozené číslo M
Výstup: Existuje ohodnocení proměnných, které splní alespoň M klausulí formule φ?

Věta 3.4.9 *Problém MAX-2-SAT je* NP*-úplný.*

Důkaz: Uvedeme dva důkazy. První důkaz je založen na p-převoditelnosti problému 3-SAT na MAX-2-SAT. V problému 3-SAT připouštíme pouze klausule délky 3. Uvažme libovolnou instanci $c_1, \ldots, c_m$ problému 3-SAT. Každou klausuli c_i nahradíme množinou 10 klausulí c'_i. Klausule c'_i obsahují literály c_i a navíc novou proměnnou w_i. Pokud $c_i = (l_1, l_2, l_3)$, pak c'_i má tvar

$$\begin{gathered}(l_1),\ (l_2),\ (l_3),\ (w_i),\\ (\neg l_1 \vee \neg l_2),\ (\neg l_1 \vee \neg l_3),\ (\neg l_2 \vee \neg l_3),\\ (l_1 \vee \neg w_i),\ (l_2 \vee \neg w_i),\ (l_3 \vee \neg w_i),\end{gathered}$$

Protože počet splněných klausulí této množiny se nezmění, pokud provedeme permutaci hodnot literálů l_1, l_2, l_3 stačí sledovat pouze to, kolik z těchto literálů je splněno a kolik nesplněno. Počet splněných klausulí zapišme do tabulky v závislosti na hodnotě w_i a $\sum_{i=1}^{3} l_i$.

		w	
		0	1
$\sum_{i=1}^{3} l_i$	0	6	4
	1	7	6
	2	7	7
	3	6	7

Z tabulky je zřejmé, že pro každé ohodnocení proměnných původní formule lze volbou hodnoty w_i dosáhnout alespoň 6 a nejvýše 7 splněných klausulí v c'_i. Navíc, 7 splněných klausulí v c'_i lze dosáhnout právě tehdy, když původní ohodnocení splňuje klausuli c_i.

Můžeme tedy učinit následující závěr. Formule $c'_1, \ldots, c'_m$ obsahuje celkem $10m$ klausulí. Pokud je původní formule $c_1, \ldots, c_m$ splnitelná, existuje ohodnocení formule $c'_1, \ldots, c'_m$, které splňuje $7m$ klausulí. Naopak, pokud lze v nové

formuli splnit $7m$ klausulí, pak musí v každé c'_i být splněno 7 klausulí a tedy vypuštěním proměnných w_i dostaneme splňující ohodnocení všech klausulí c_i. Původní formule je tedy splnitelná.

Zobrazení f definované jako $f(c_1, \ldots, c_m) = \langle (c'_1, \ldots, c'_m), 7m \rangle$ je tedy p-převoditelností problému 3-SAT na MAX-2-SAT.

Ve druhém důkazu předvedeme p-převoditelnost problému kliky na problém MAX-2-SAT. Nechť $\langle (V, E), k \rangle$ je instance problému kliky. Předpokládejme, že $V = \{v_1, \ldots, v_n\}$. Uzlu v_i přiřadíme proměnnou x_i. Do formule zařadíme klausule (x_i) pro všechna $i = 1, \ldots, n$ a klausule $(\neg x_i \vee \neg x_j)$ pro každou dvojici i, j, pro kterou $(v_i, v_j) \notin E$. První skupině klausulí budeme říkat vrcholové klausule a druhé skupině nehranové klausule. Instanci problému MAX-2-SAT dostaneme tak, že požadavek na počet současně splnitelných klausulí zvolíme $\binom{n}{2} - |E| + k$.

Tvrdíme, že každé ohodnocení proměnných získané formule lze upravit tak, že všechny nehranové klausule budou splněny a celkový počet splněných klausulí neklesne. Úpravu provedeme následovně. Pokud je některá nehranová klausule nesplněna, vybereme jeden z jejích vrcholů a jeho ohodnocení změníme na 0. Tím ubyde právě jedna splněná vrcholová klausule, ale přibude alespoň jedna nehranová klausule.

Z dokázané vlastnosti ohodnocení plyne, že při hledání maximálního počtu splněných klausulí se stačí omezit na ohodnocení, která splňují všechny nehranové klausule. To nastane právě tehdy, když každé dva vrcholy ohodnocené 1 jsou spojeny hranou a neexistuje nehranová klausule, do které by oba patřily. Lze se tedy omezit na hledání ohodnocení, která definují kliku. Pro porovnání počtu splněných klausulí pro dvě kliky hraje roli pouze počet splněných vrcholových klausulí, který je roven velikosti kliky. Popsané zobrazení tedy určuje p-převoditelnost problému kliky na MAX-2-SAT. □

4 Polynomiální algoritmy pro modifikace NP-úplných úloh

4.1 Algoritmus pro problém batohu v jednotkovém kódování vstupu

Jednotkové kódování vstupu v podstatě znamená, že za délku vstupu považujeme součet velikosti vstupních čísel. Přesně to znamená, že za délku vstupu považujeme délku, kterou by zápis vstupu měl, kdybychom v něm čísla kódovali tak, že kódem čísla n je slovo 1^n. Algoritmus, který pracuje v čase polynomiálním od takto definované délky vstupu má pochopitelně exponenciální čas výpočtu, pokud bychom vstup kódovali standardním způsobem pomocí binární číselné soustavy.

Předpokládejme na vstupu instanci $a_1, \ldots, a_n, b$. Algoritmus postupně pro každé $k = 0, \ldots, n$ zkonstruuje množinu

$$S_k = \left\{ \sum_{i \in I} a_i;\ I \subseteq \{1, \ldots, k\} \right\} \cap [0, b]$$

Ke konstrukci těchto množin se využijí vztahy

$$\begin{aligned} S_0 &= \{0\} \\ S_k &= S_{k-1} \cup \{s + a_k; s \in S_{k-1}\} \cap [0, b] \end{aligned}$$

Vstupní instance problému batohu má řešení právě tehdy, když $b \in S_n$.

Konstrukci množin S_k i test $b \in S_n$ lze provést v polynomiálním čase od $\sum_{i=1}^{n} a_i + b$.

4.2 Algoritmus pro 2-SAT

Dokážeme o něco obecnější výsledek, protože budeme uvažovat (≤ 2)-SAT místo 2-SAT, t.j. budeme se zabývat problémem splnitelnosti pro formule s klausulemi délky nejvýše 2.

Název: Problém (≤ 2)-SAT
Vstup: Booleovská formule v CNF, která obsahuje nejvýše dva literály v každé klausuli.
Výstup: Je formule splnitelná?

Věta 4.2.1 *Pro úlohu (≤ 2)-SAT existuje polynomiální algoritmus.*

Důkaz: Použijeme úplnost rezoluční metody dokazování. Připomeňme, že v jednom kroku rezoluce odvodíme ze dvou klausulí, které obsahují právě jednu dvojici komplementárních literálů, novou klausuli následujícím způsobem.

$$\frac{(a_1 \vee a_2 \vee \ldots \vee a_k \vee c), (b_1 \vee b_2 \vee \ldots \vee b_l \vee \neg c)}{(a_1 \vee a_2 \vee \ldots \vee a_k \vee b_1 \vee b_2 \vee \ldots \vee b_l)}$$

Jinak řečeno, komplementární literály vypustíme a zbylé části klausulí spojíme do jedné. Platí následující věta.

Věta 4.2.2 *Množina klausulí je splnitelná právě tehdy, když z ní nelze odvodit spor, tj. klausuli neobsahující žádný literál.*

Důkaz: Jestliže je z množiny klausulí odvoditelný spor, pak zřejmě není splnitelná. Nechť naopak není nějaká množina klausulí splnitelná. Dokážeme, že z ní lze odvodit spor. Předpokládejme, že uvažovaná množina klausulí je na množině proměnných $x_1, x_2, \ldots, x_n$.

Budeme používat částečná ohodnocení proměnných. Tím budeme rozumět ohodnocení proměnných $x_1, x_2, \ldots, x_i$ pro libovolné $i = 0, 1, \ldots, n$. Speciálně, při $i = 0$ máme prázdné ohodnocení, které nepřiřazuje hodnotu žádné proměnné.

Uspořádejme částečná ohodnocení do stromu, jehož kořen je prázdné ohodnocení. Na hladině i budou právě všechna ohodnocení prvních i proměnných. Každé ohodnocení na hladině $i \leq n-1$ má dva následníky, které rozšiřují dané ohodnocení o obě možné varianty ohodnocení proměnné x_{i+1}.

Libovolné ohodnocení v uvedeném stromě, které přiřazuje všem literálům některé klausule hodnotu nepravda, nazveme uzavřené. Protože všechny literály dané klausule jsou již ohodnoceny, nemůže ohodnocení dalších proměnných změnit její ohodnocení.

Pokud spor, tj. prázdná klausule, není přímo prvkem uvažované množiny klausulí, pak prázdné ohodnocení, které je přiřazeno kořeni stromu, není uzavřené. Ohodnocení v listech stromu přiřazují hodnoty všem proměnným. Protože jsme vyšli z předpokladu, že uvažovaná množina klausulí je nesplnitelná, jsou všechna tato ohodnocení uzavřená.

Ukážeme, že ve stromu existuje ohodnocení, které není uzavřené, ale oba jeho následníci jsou uzavřená ohodnocení. Takové ohodnocení lze nalézt například tak, že vyjdeme z kořene a kdykoli navštívíme uzel, jehož některý následník není uzavřený, přejdeme do tohoto následníka. Tento postup skončí nejpozději na hladině $n-1$, protože na hladině n jsou již všechna ohodnocení uzavřená.

Vezměme tedy některé otevřené ohodnocení u, jehož následníci doplňují ohodnocení nějaké proměnné x_i a obě možnosti dávají uzavřené ohodnocení. Následník, který doplňuje $x_i = \text{true}$ nesplňuje nějakou klausuli. Ta musí obsahovat literál $\neg x_i$. Nechť je to klausule $(a_1 \vee a_2 \vee \ldots \vee a_k \vee \neg x_i)$. Analogicky, následník, který doplňuje $x_i = \text{false}$ nesplňuje některou klausuli, která obsahuje literál x_i. Nechť je to klausule $(b_1 \vee b_2 \vee \ldots \vee b_l \vee x_i)$. Protože všechny literály uvedených dvou klausulí jsou v příslušných následnících u nesplněny, musí literály $a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_l$ být nesplněny již v ohodnocení u. Nyní přidejme do uvažované množiny klausulí rezoluční důsledek uvedených dvou klausulí, tj. klausuli $(a_1 \vee a_2 \vee \ldots \vee a_k \vee b_1 \vee b_2 \vee \ldots \vee b_l)$. Pro takto rozšířenou množinu klausulí se ohodnocení u stane uzavřené, protože v přidané klausuli ohodnocuje všechny literály jako nepravda.

Opakujeme-li uvedený postup rozšiřování množiny klausulí o rezoluční důsledky, snížíme v každém kroku počet otevřených ohodnocení aspoň o jedno. Po konečném počtu kroků tedy dostaneme množinu klausulí, pro níž je uzavřené již ohodnocení v kořeni stromu. To může nastat jen v tom případě, že získaná množina klausulí obsahuje spor.

Dokázali jsme tedy, že z libovolné nesplnitelné množiny klausulí je odvoditelný spor. Tím je pomocná věta dokázána. □

Nyní můžeme dokončit důkaz polynomiální řešitelnosti problému 2-SAT tím, že popíšeme polynomiální algoritmus na její řešení. Algoritmus systematicky probírá všechny dvojice klausulí a zjišťuje jejich rezoluční důsledky. Trik algoritmu je v tom, že v případě 2-SAT zůstává délka rezolučních důsledků nejvýše 2. Aby se v algoritmu zabránilo opakovanému probírání téže dvojice klausulí, bude prohledávání uspořádáno následovně. Množina klausulí D bude obsahovat všechny dosud nalezené rezoluční důsledky. Pokud se najde rezoluční důsledek klausulí v D, který ještě není v D, je do D přidán. Kromě toho budeme konstruovat množinu $Z \subseteq D$ (zpracované klausule), pro kterou bude v každém okamžiku platit, že pro každou dvojici klausulí ze Z již byly zkonstruovány všechny jejich rezoluční důsledky a zařazeny do D. Jednoprvková množina klausulí tuto podmínku splňuje, proto Z může být inicializována jako $\{c_1\}$. Dvojice klausulí, které obě patří do Z již nebudou znovu probírány. Jestliže již nelze přidat žádný nový důsledek, algoritmus končí. Odpověď algoritmu je pak určena tím, zda v množině důsledků D je nebo není sporná klausule. Podrobný algoritmus je následující.


```
TestSplnitelnostiResolution(c_1, ..., c_m) {
    D ← {c_1, ..., c_m};
    Z ← {c_1};
    while (Z ≠ D) {
        Zvol libovolně c ∈ D \ Z.
        Pro každou c' ∈ Z zkonstruuj rezoluční důsledek
        klausulí c, c', pokud je definován, a přidej jej
        do D, pokud tam ještě není.
        Z ← Z ∪ {c};
    }
    if (D obsahuje spornou klausuli) return("formule je nesplnitelná");
    else return("formule je splnitelná");
}
```

Protože všechny vstupní klausule mají délku nejvýše 2, mají i všechny jejich rezoluční důsledky délku nejvýše 2. V klausuli nepřipouštíme přítomnost dvou literálů obsahujících stejnou proměnnou. V každém okamžiku tedy platí $|D| \leq 4\binom{n}{2}+2n+1$, kde jednotlivé sčítance odpovídají počtu klausulí se dvěma, jedním a žádným literálem. Protože se množina Z v každém kroku algoritmu zvětší, bude nejpozději po $4\binom{n}{2}+2n+1$ krocích splněno $Z = D$ a algoritmus skončí.

Množina klausulí získaná algoritmem obsahuje všechny rezoluční důsledky vstupní množiny klausulí, a proto je možné získat odpověď na původní otázku. Vstupní množina klausulí je splnitelná právě tehdy, když v získané množině rezolučních důsledků není spor. □

Polynomiální algoritmus pro problém 2-SAT lze odvodit také pomocí *jednotkové propagace* (unit propagation), což je následující postup. Ve formuli tvaru CNF se vyhledají klauzule délky 1, za proměnné v těchto klauzulích se dosadí hodnoty, které tyto klauzule splní, a tento postup se opakuje, pokud ve formuli vznikly nové klauzule délky 1. Postup končí, pokud se ve formuli objeví prázdná klauzule nebo všechny klauzule ve formuli mají délku alespoň 2. Pro vstupní formuli φ budeme výslednou formuli značit $\text{unitprop}(\varphi)$. Formule φ a $\text{unitprop}(\varphi)$ jsou ekvisplnitelné, tedy φ je splnitelná právě tehdy, když je splnitelná $\text{unitprop}(\varphi)$. Speciálně, pokud $\text{unitprop}(\varphi)$ obsahuje prázdnou klauzuli, je φ nesplnitelná a pokud je $\text{unitprop}(\varphi)$ prázdná, je φ splnitelná.

Formuli, která vznikne z φ dosazením $a \in \{0,1\}$ za proměnnou x, budeme značit $\varphi[x = a]$.

Dosazení za proměnné ve formuli tvaru CNF nazveme *autark assignment*, pokud splní každou klauzuli, ve které ohodnotí některý literál. Lze snadno ověřit následující. Jestliže φ je formule a ψ vznikne z φ odstraněním klauzulí, které jsou splněny některým autark assignment, je ψ ekvisplnitelná φ. K důkazu algoritmu z Obrázku 2 je potřeba ověřit následující.

Lemma 4.2.3 *Jestliže φ je libovolná instance problému 2-SAT, $\varphi_0 = \text{unitprop}(\varphi[x = 0])$ a $\varphi_1 = \text{unitprop}(\varphi[x = 1])$, pak platí následující. Pokud obě formule φ_0 a φ_1 obsahují prázdnou klauzuli,*

```
TestSplnitelnostiUP(φ) {
    φ ← unitprop(φ)
    if (φ obsahuje spornou klausuli) return("formule je nesplnitelná")
    while (φ není prázdná formule) {
        zvol libovolnou proměnnou x ve φ.
        ψ ← unitprop(φ[x = 0])
        if (ψ obsahuje spornou klausuli) {
            ψ ← unitprop(φ[x = 1])
        }
        if (ψ obsahuje spornou klausuli) return("formule je nesplnitelná")
        φ ← ψ
    }
    return("formule je splnitelná");
}
```

Obrázek 2: Algoritmus pro 2-SAT

pak je φ nesplnitelná. Pokud kterákoli z formulí φ_0 a φ_1 neobsahuje prázdnou klauzuli, pak je ekvisplnitelná s formulí φ.

Důkaz: Jestliže φ_i pro některé $i = 0, 1$ neobsahuje prázdnou klauzuli, lze ověřit, že φ_i vznikne z φ pomocí autark assignment, protože všechny klauzule, ve kterých je ohodnocen některý literál, jsou buď tímto literálem splněny, nebo se změní na klauzule délky 1 a jsou v dalším kroku splněny nebo se změní na prázdnou klauzuli. Z toho plyne, že φ_i je ekvisplnitelná φ. □

5 Aproximační algoritmy

5.1 Aproximační algoritmus pro problém vrcholového pokrytí

Popíšeme algoritmus, který pro libovolný vstupní graf $G = (V, E)$ vydá jako výstup množinu vrcholů $A \subseteq V$, která je vrcholovým pokrytím a velikost $|A|$ je nejvýše dvakrát větší než velikost nejmenšího vrcholového pokrytí. Úloha najít přesné minimální vrcholové pokrytí je NP-úplná.

Vstup: Graf $G = (V, E)$.
Výstup: Aproximace nejmenšího vrcholového pokrytí.
Seřaď hrany G v libovolném pořadí.
$A \leftarrow \emptyset$;
Probírej postupně všechny hrany ve zvoleném pořadí: {
    Pro každou hranu (u, v) proveď
        if $(\{u, v\} \cap A = \emptyset)$ then $A \leftarrow A \cup \{u, v\}$
}
return(A);

Věta 5.1.1 *Výstup uvedeného algoritmu je vrcholové pokrytí velikosti nejvýše dvakrát větší než je velikost nejmenšího vrcholového pokrytí.*

Důkaz: Algoritmus zaručuje, že z každé hrany obsahuje výstupní množina A aspoň jeden vrchol. Je to tedy vrcholové pokrytí.

Nechť B je libovolné minimální vrcholové pokrytí daného grafu. Uvažme nyní některý krok, kdy algoritmus přidal nové vrcholy u, v do A. Protože B je vrcholové pokrytí, aspoň jeden z těchto vrcholů je prvkem B. Z toho plyne, že v průběhu algoritmu nastane nejvýše $|B|$ kroků, kdy algoritmus přidal nové vrcholy. Protože v jednom takovém kroku přidá algoritmus dva nové vrcholy, je $|A| \leq 2|B|$. □

5.2 Aproximace problému obchodního cestujícího

Nejprve si popíšeme problém samotný.

Název: Problém obchodního cestujícího
Vstup: Množina n uzlů označených čísly $1, 2, \ldots, n$ a "vzdálenost" $d_{i,j}$ mezi uzly i a j pro každé $i \neq j$. Předpokládáme, že vzdálenosti splňují pro každé i, j, k trojúhelníkovou nerovnost $d_{i,j} \leq d_{i,k} + d_{k,j}$ a že $d_{i,j} = d_{j,i}$.
Výstup: Některá z permutací $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$, pro kterou je

$$\sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)}$$

minimální.

Uvedeme si polynomiální algoritmus, který nalezne aproximaci s faktorem $\alpha = 3/2$. K tomu budeme potřebovat následující pojmy a tvrzení.

Definice 5.2.1 Graf nazveme Eulerovský, jestliže je souvislý a všechny jeho vrcholy mají sudý stupeň.

Definice 5.2.2 Graf nazveme párování, jestliže v něm žádné dvě hrany nemají společný vrchol. Párování nazveme úplné, jestliže má graf sudý počet vrcholů a z každého vrcholu v něm vede hrana.

Definice 5.2.3 Kostrou souvislého grafu $G = (V, E)$ nazveme strom (souvislý graf bez cyklů), který obsahuje všechny vrcholy V.

Platí následující tři věty, které nebudeme dokazovat. Sledem v grafu se rozumí posloupnost navazujících hran. Váženým grafem rozumíme graf, jehož hrany jsou ohodnoceny kladnými racionálními čísly.

Věta 5.2.4 *V každém Eulerovském grafu, který případně může obsahovat násobné hrany, existuje uzavřený sled, který prochází všemi hranami.*

Tuto větu lze také zformulovat tak, že každý Eulerovský graf lze nakreslit jedním uzavřeným tahem.

Věta 5.2.5 *Existuje polynomiální algoritmus, který v libovolném váženém grafu nalezne některé z úplných párování, které má minimální součet vah na hranách.*

Věta 5.2.6 *Existuje polynomiální algoritmus, který v libovolném váženém grafu nalezne kostru, která má minimální součet vah na hranách.*

Nyní popíšeme aproximační algoritmus pro problém obchodního cestujícího.

Na základě vstupní matice čísel $\{d_{i,j}\}_{i,j=1}^{n}$ vytvoř úplný hranově ohodnocený graf G na n vrcholech.
Nalezni kostru grafu G minimální váhy a označ ji K.
Nechť H je množina vrcholů, které mají v K lichý stupeň.
Nalezni minimální párování P restrikce grafu G na množinu H.
Nechť $W = K \cup P$, přičemž společné hrany K a P budou ve W dvakrát.
Graf W (případně s násobnými hranami) je Eulerovský.
Nechť S je uzavřený sled ve W, který prochází všemi hranami.
Ze sledu S budeme vypouštět uzly, kterými sled prochází opakovaně, tak dlouho, až získáme uzavřený sled S', který prochází každým uzlem právě jednou.
Sled S' je uzavřenou cestou v G.
Pořadí vrcholů, ve kterém nalezená cesta graf prochází, určuje permutaci, která je výstupem algoritmu.

Věta 5.2.7 *Délka cesty S' nalezené algoritmem je nejvýše $3/2$ krát větší než délka optimální cesty.*

Důkaz: Nechť C je některá optimální cesta v G a d její délka. Vypuštěním kterékoli hrany z ní získáme kostru. Kostra minimální váhy K má tedy váhu nejvýše d. Postupným vypouštěním uzlů z C, které nepatří do H, dostaneme s využitím trojúhelníkové nerovnosti uzavřenou cestu C', která prochází pouze přes vrcholy H a jejíž délka je nejvýše d. Lze snadno ověřit, že H má sudý počet prvků. Rozdělením C' na sudé a liché hrany tedy získáme dvě párování na H, jejichž váhy jsou dohromady nejvýše d. Alespoň jedno z těchto párování tedy má váhu nejvýše $d/2$.

Graf K' vznikl jako sjednocení minimální kostry váhy nejvýše d a minimálního párování váhy nejvýše $d/2$. Sled, který algoritmus nalezne v K', tedy má váhu nejvýše $3/2 \cdot d$ a stejná mez platí i pro výslednou cestu S'. □

5.3 Aproximační algoritmus pro množinové pokrytí

Nejprve popíšeme úlohu samotnou.

Název: Množinové pokrytí
Vstup: Konečné množiny $S_1, \ldots, S_n$
Výstup: Najít minimální $C \subseteq \{1, \ldots, n\}$ tak, že

$$\bigcup_{i \in C} S_i = \bigcup_{i=1}^{n} S_i.$$

Nechť $m = |\bigcup_{i=1}^{n} S_i|$. Budeme analyzovat následující algoritmus.

for $(i = 1, \ldots, n)$ $U_i \leftarrow S_i$
$C \leftarrow \emptyset$
while (některá z množin U_i je neprázdná) {
  Nechť j je index některé z největších množin U_i.
  $C \leftarrow C \cup \{j\}$
  for $(i = 1, \ldots, n)$ $U_i \leftarrow U_i \setminus S_j$
}
return(C)

Věta 5.3.1 *Nechť C^* je některé optimální pokrytí a C^A pokrytí nalezené algoritmem. Pokud $|C^*| \geq 3$, pak $|C^A| \leq |C^*| \ln m$.*

Důkaz: Nechť $C^* = \{i_1, \ldots, i_r\}$. Pro každé $k = 0, \ldots, |C^A|$ označme $X_k = \bigcup_{i=1}^{n} U_i$ v okamžiku po přidání k-tého prvku do C a odečtení příslušné S_j. Speciálně, $X_0 = \bigcup_{i=1}^{n} U_i$ před zahájením cyklu, tedy $X_0 = \bigcup_{i=1}^{n} S_i$.

Protože $S_{i_1} \cup \ldots \cup S_{i_r} = \bigcup_{i=1}^{n} S_i$ a všechny množiny U_i dostaneme z odpovídajících S_i odečtením stejných množin S_j vybraných až do daného kroku, platí $U_{i_1} \cup \ldots \cup U_{i_r} = \bigcup_{i=1}^{n} U_i = X_k$. Mezi množinami $U_{i_1}, \ldots, U_{i_r}$ tedy existuje množina velikosti alespoň $\frac{1}{r}|X_k|$. Index j je v každém kroku vybrán tak, aby množina $|U_j|$ měla maximální velikost mezi všemi množinami $U_1, \ldots, U_n$ a tedy je alespoň tak veliká jako největší z množin $U_{i_1}, \ldots, U_{i_r}$. Platí tedy $|U_j| \geq \frac{1}{r}|X_k|$. Protože $|X_{k+1}| = |X_k \setminus S_j| = |X_k \setminus U_j| = |X_k| - |U_j|$, platí také $|X_{k+1}| \leq \left(1 - \frac{1}{r}\right)|X_k|$. Protože $|X_0| = m$, dostaneme indukcí pro $k \geq 0$

$$|X_k| \leq \left(1 - \frac{1}{r}\right)^k m \ .$$

Dále, $|C^A|$ je rovno minimálnímu k, pro které je $|X_k| = 0$. Označme $k_0 = \lceil r \ln(m/2) \rceil$. Protože $k_0 \geq r \ln(m/2)$, dostaneme

$$|X_{k_0}| \leq \left(1 - \frac{1}{r}\right)^{k_0} m \leq \left(1 - \frac{1}{r}\right)^{r \ln(m/2)} m \ .$$

S využitím nerovnosti $1 - \frac{1}{r} < \mathrm{e}^{-1/r}$ pro libovolné $r > 0$ z toho dále plyne

$$|X_{k_0}| < \mathrm{e}^{-\ln(m/2)} m = \frac{m}{m/2} = 2$$

Protože $|X_{k_0}|$ je přirozené číslo, je $|X_{k_0}| \leq 1$. V každém kroku algoritmu se X_k změnší alespoň o jeden prvek. Po odečtení nejvýše $k_0 + 1$ množin S_j jsou tedy všechny U_i prázdné, a je tedy $|C^A| \leq k_0 + 1$.

Platí $k_0 + 1 = \lceil r \ln m - r \ln 2 + 1 \rceil$. Protože $r = |C^*| \geq 3$, platí $r \ln 2 > 2$, a tedy $k_0 + 1 \leq \lceil r \ln m - 1 \rceil \leq r \ln m$. Celkem tedy platí $|C^A| \leq |C^*| \ln m$. □

K formulaci dalšího odhadu zaveďme následující značení pro částečné součty harmonické řady.

Definice 5.3.2 Pro libovolné přirozené číslo $s \geq 1$ nechť

$$\mathcal{H}(s) = \sum_{k=1}^{s} \frac{1}{k} .$$

Je známo, že platí

$$\mathcal{H}(s) = \sum_{k=1}^{s} \frac{1}{k} = \ln s + \gamma + o(1) .$$

kde $\gamma = 0.577215...$ je Eulerova konstanta.

Platí následující odhad pro $|C^A|$, který závisí na velikosti největší ze vstupních množin $|S_i|$ a nikoli na velikosti jejich sjednocení m.

Věta 5.3.3 *Nechť C^* je některé optimální pokrytí a C^A pokrytí nalezené výše popsaným algoritmem. Pak platí $|C^A| \leq \mathcal{H}(\max_i |S_i|)\, |C^*|$.*

Důkaz: Uvažme situaci po ukončení algoritmu. Označme $k = |C^A|$ a přečíslujme množiny S_i tak, aby pro $i = 1, \ldots, k$ byla v i-tém kroku vybrána množina S_i. Označme ještě $F = \{S_1, \ldots, S_n\}$ a $X = \bigcup_{i=1}^{n} S_i$.

Nechť $x \in X$ a nechť je prvek x poprvé pokryt v kroku j. Pak definujme

$$c_x = \frac{1}{|S_j \setminus (S_1 \cup \ldots \cup S_{j-1})|} .$$

Protože v j-tém kroku je poprvé pokryto právě $|S_j \setminus (S_1 \cup \ldots \cup S_{j-1})|$ prvků, je součet c_x přes prvky pokryté poprvé v témže kroku roven 1. Platí tedy

$$\sum_x c_x = |C^A| .$$

Následující nerovnost platí proto, že každý prvek $x \in X$ přispívá do levé strany právě c_x a do pravé strany c_x vynásobené počtem množin C^*, které pokrývají daný prvek, tedy alespoň c_x.

$$\sum_x c_x \leq \sum_{j \in C^*} \sum_{x \in S_j} c_x .$$

Tvrdíme, že k důkazu věty stačí dokázat následující tvrzení.

Tvrzení 5.3.4 *Pro každé $S \in F$ platí*

$$\sum_{x \in S} c_x \leq \mathcal{H}(|S|) .$$

Pokud toto tvrzení platí, pak platí také

$$\sum_x c_x \leq \sum_{j \in C^*} \sum_{x \in S_j} c_x \leq \sum_{j \in C^*} \mathcal{H}(|S_j|) \leq \sum_{j \in C^*} \mathcal{H}(\max_i |S_i|) \leq |C^*| \mathcal{H}(\max_i |S_i|) ,$$

což dokazuje tvrzení věty. Zbývá tedy dokázat Tvrzení 5.3.4.

Důkaz: Pro danou $S \in F$ a libovolné $i = 1, \ldots, k$ nechť

$$u_i(S) = |S \setminus (S_1 \cup \ldots \cup S_i)| ,$$

speciálně

$$u_0(S) = |S|$$

a

$$u_k(S) = 0$$

a k je nejmenší index i pro který je $u_i(S) = 0$. V i-tém kroku je z množiny S poprvé pokryto právě $u_{i-1}(S) - u_i(S)$ prvků. Platí tedy

$$\sum_{x \in S} c_x = \sum_{i=1}^{k} (u_{i-1}(S) - u_i(S)) \cdot \frac{1}{|S_i \setminus (S_1 \cup \ldots \cup S_{i-1})|} .$$

Kdyby $u_{i-1}(S) > |S_i \setminus (S_1 \cup \ldots \cup S_{i-1})|$, byla by v i-tém kroku vybrána S místo S_i. Platí tedy

$$u_{i-1}(S) \leq |S_i \setminus (S_1 \cup \ldots \cup S_{i-1})| ,$$

což spolu s předchozím implikuje

$$\sum_{x \in S} c_x \leq \sum_{i=1}^{k} (u_{i-1}(S) - u_i(S)) \cdot \frac{1}{u_{i-1}(S)} .$$

Substitucí $i = k - j + 1$ pro $j = 1, \ldots, k$ dostaneme

$$\sum_{x \in S} c_x \leq \sum_{j=1}^{k} (u_{k-j}(S) - u_{k-j+1}(S)) \cdot \frac{1}{u_{k-j}(S)} . \tag{9}$$

Přepišme tuto sumu na součet posloupnosti, která bude složena z opakovaných výskytů zlomků $\frac{1}{u_{i-1}(S)}$ pro $i = k, \ldots, 1$ podle následující tabulky.

zlomek	počet opakování
$\frac{1}{u_{k-1}(S)}$	$u_{k-1}(S) - u_k(S)$
$\ldots$	$\ldots$
$\frac{1}{u_{k-j}(S)}$	$u_{k-j}(S) - u_{k-j+1}(S)$
$\ldots$	$\ldots$
$\frac{1}{u_0(S)}$	$u_0(S) - u_1(S)$

Posloupnost se skládá z bloků tvořených opakováním téhož zlomku a její součet je roven pravé straně (9). Počet bloků je k a celkovou délku posloupnosti vypočteme jako součet délek bloků od posledního k prvnímu

$$(u_0(S) - u_1(S)) + \ldots + (u_{k-j}(S) - u_{k-j+1}(S)) + \ldots + (u_{k-1}(S) - u_k(S)) \\ = u_0(S) - u_k(S) = |S| \ .$$

Uvažovaná posloupnost má tedy stejnou délku jako posloupnost

$$\frac{1}{1}, \frac{1}{2}, \ldots, \frac{1}{|S|} \ . \tag{10}$$

Ukážeme, že členy posloupnosti (10) omezují shora odpovídající členy uvažované posloupnosti zlomků $\frac{1}{u_{k-j}(S)}$. První blok má délku $u_{k-1}(S) - u_k(S) = u_{k-1}(S)$ a skládá se z opakování zlomku $\frac{1}{u_{k-1}(S)}$. Jeho poslední člen se tedy rovná odpovídajícímu členu posloupnosti (10) a předchozí členy jsou menší. Obecně, prvních j bloků má v součtu délku $u_{k-j}(S)$ a j-tý blok se skládá z opakování zlomku $\frac{1}{u_{k-j}(S)}$. Poslední člen j-tého bloku se tedy opět rovná odpovídajícímu členu (10) a předchozí členy jsou menší. Spolu s (9) to dokončuje důkaz tvrzení a tedy také věty. □ □

6 PSPACE

6.1 Základní pojmy a vlastnosti

Třída PSPACE je třídou úloh, které lze na TS řešit v polynomiálním prostoru. Formální definice je uvedena v Sekci 2.3.

Věta 6.1.1 NP $\subseteq$ PSPACE.

Důkaz: Pro každý jazyk $L \in$ NP v libovolné abecedě Γ_1 existuje p-omezená relace R v P, abeceda Γ_2 a polynom $p(n)$ tak, že pro každé slovo $w \in \Gamma_1^*$ platí

$$w \in L \iff (\exists x \in \Gamma_2^*)\ (w\#x \in R)$$

a

$$(\forall x \in \Gamma_2^*)\ (w\#x \in R \Rightarrow |x| \leq p(|w|)).$$

Testování, zda $w\#x \in R$ lze provést v polynomiálním čase a tedy i v polynomiálním prostoru. Protože tyto testy jsou pro různá x nezávislé, je možné po provedení každého z nich použitou oblast pásky uvolnit a použít pro další test. Informace, které je třeba zachovávat po dobu celého výpočtu, zahrnují pouze omezení délky slov x, záznam posledního testovaného slova x a informaci, zda již bylo nebo nebylo nalezeno slovo x, splňující $w\#x \in R$. Prostor pro průběžné záznamy i pro jednotlivé testy je polynomiální, proto stačí polynomiální prostor pro celý výpočet. □

Třída NP patří do polynomiální hierarchie (PH), protože NP $= \Sigma_1^P$. Důkaz předchozí věty lze snadno zobecnit a dokázat, že třída PSPACE obsahuje všechny třídy PH, tedy Σ_k^P a Π_k^P pro $k \geq 1$. Na druhé straně, pokud předpokládáme ostrou inkluzi $\Sigma_k^P \subset \Sigma_{k+1}^P$ pro všechna $k \geq 1$ nebo, ekvivalentně, $\Pi_k^P \subset \Pi_{k+1}^P$ pro všechna $k \geq 1$, pak lze dokázat, že PSPACE obsahuje i úlohy, které nepatří do PH.

Definice 6.1.2 *Jazyk L nazveme* PSPACE*-úplný, pokud*
1. $L \in$ PSPACE.
2. Každý $L' \in$ PSPACE *je p-převoditelný na* L.

Úlohy, které jsou PSPACE-úplné, jsou nejtěžšími úlohami ve třídě PSPACE. Pokud by existoval polynomální algoritmus pro kteroukoli z nich, existoval by polynomiální algoritmus pro všechny úlohy z PSPACE.

6.2 PSPACE úplnost problému QBF

Kvantifikovanou Booleovskou formulí budeme rozumět formuli, která kromě operací obvyklých v Booleovských formulích může obsahovat kvantifikaci Booleovských proměnných. Kvantifikované Booleovské formule budeme uvažovat pouze v prenexním tvaru, tedy ve tvaru $(Q_1 x_1) \ldots (Q_n x_n)\, \varphi$, kde pro $i = 1, \ldots, n$ je Q_i některý kvantifikátor a φ je formule bez kvantifikátorů, kterou budeme nazývat vnitřní část kvantifikované Booleovské formule. Pravdivost uzavřené formule tohoto tvaru je definována obvyklým významem kvantifikátorů, který je v tomto případě vyjádřen vztahy

$$\begin{aligned} (\forall x)\varphi(x) &\equiv \varphi(0) \wedge \varphi(1), \\ (\exists x)\varphi(x) &\equiv \varphi(0) \vee \varphi(1). \end{aligned}$$

Příklad. Kvantifikovaná Booleovská formule $(\forall x)(\forall y)((x \vee \neg y) \wedge (\neg x \vee y))$ není pravdivá, protože vnitřní část formule není splněna pro $x = 0$, $y = 1$. Na druhé straně, formule $(\forall x)(\exists y)((x \vee \neg y) \wedge (\neg x \vee y))$ je pravdivá, protože pro každou hodnotu x je vnitřní část formule splněna pro $y = x$.

Kvantifikované Booleovské formule lze po úpravě syntaxe interpretovat v predikátovém počtu. Zvolíme jazyk, který obsahuje konstanty 0, 1 a predikát $p(x)$. Kvantifikovanou Booleovskou formuli přepíšeme do predikátového počtu s tímto jazykem tak, že ve vnitřku formule nahradíme každou proměnnou x atomickou formulí $p(x)$. Například první formule z příkladu výše bude přepsána do tvaru $(\forall x)(\forall y)((p(x) \vee \neg p(y)) \wedge (\neg p(x) \vee p(y)))$. Pravdivost uzavřených kvantifikovaných Booleovských formulí je pak ekvivalentní pravdivosti jejich přepisu do predikátového počtu v modelu, který je určen axiomy

$$\begin{aligned} &(\forall x)(x = 0 \vee x = 1)\ , \\ &\neg p(0)\ , \\ &p(1)\ . \end{aligned}$$

Uzavřená kvantifikovaná Booleovská formule je v normálním tvaru, pokud je v prenexním tvaru a její vnitřní část je v CNF nebo DNF. Problémem QBF budeme rozumět následující úlohu.

Název: Pravdivost kvantifikovaných Booleovských formulí (QBF).
Vstup: Uzavřená kvantifikovaná Booleovská formule v normálním tvaru.
Výstup: Je vstupní formule pravdivá?

Věta 6.2.1 *Problém* QBF *patří do třídy* PSPACE.

Důkaz: Vyhodnocení pravdivosti formulí tvaru QBF lze provést rekurzivní procedurou na Obrázku 3. Pokud vstupní formule φ není konstantní výraz, předpokládá se, že má tvar $\varphi = (Q\,x)\psi(x)$, kde Q je kvantifikátor. Dále je použito značení $\mathrm{op}(Q)$ definované tak, že $\mathrm{op}(Q) = \vee$, pokud $Q = \exists$, a $\mathrm{op}(Q) = \wedge$, pokud $Q = \forall$.

```
function Vyhodnoť(φ) {
    if (φ je konstantní výraz) then
        return(hodnota φ);
    else   {předpokládáme φ = (Q x)ψ(x)}
        a ← Vyhodnoť(ψ(0));
        b ← Vyhodnoť(ψ(1));
        return(a op(Q) b);
    fi
}
```

Obrázek 3: Vyhodnocení formule tvaru QBF rekurzí.

Pro realizaci na TS nahradíme rekurzi pomocí cyklu následujícím způsobem. Na pásce jsou zapsány znaky ## a za ním posloupnost záznamů, z nichž každý je ukončen opět znakem #. V každém záznamu jsou zapsány hodnoty lokálních proměnných pro jedno volání rekurzivní funkce v pořadí v, a, b, φ, kde v je proměná pro uložení výsledku a a, b, φ jsou proměnné použité v rekurzivní funkci. Proměnné v, a, b jsou reprezentovány znaky z množiny $\{0, 1, \varepsilon\}$, kde ε znamená nedefinováno. Program podle potřeby vytváří záznamy pro podřízená volání rekurzivní funkce, které jsou přidány vždy na konec posloupnosti. Je-li některé volání ukončeno, je jeho záznam vymazán a výsledek je zapsán do odpovídající proměnné nadřízeného volání. Záznam pro právě aktivní volání rekurzivn9 funkce je tedy vždy poslední nebo předposlední záznam posloupnosti.

Výpočet je popsán schematem programu na Obrázku 4. Na začátku výpočtu je na pásce vytvořen záznam pro výpočet vstupní formule φ ve tvaru $\#\#\,\varepsilon\,\varepsilon\,\varepsilon\,\varphi\,\#$. Program pak pokračuje cyklem, který je ukončen provedením některého příkazu return, který určí výstupní hodnotu. Instrukcí přejít na záznam některého volání rekurzivní funkce rozumíme, že hlava TS je nastavena na jeho začátek, tedy na znak reprezentující proměnnou v tohoto volání. Proměnné v, a, b, φ v zápisu programu označují proměnné právě aktivního volání a $v_{\text{podřízené}}$ označuje výslednou hodnotu v v ukončeném podřízeném volání.


```
založit záznam pro výpočet vstupní formule φ a přejít na něj
while (true) do
    if (φ je konstantní výraz) then
        v ← hodnota(φ)
        if (existuje nadřízené volání) then
            přejít na nadřízené volání
        else
            return(v)
        fi
    {dále předpokládáme φ = (Q x)ψ(x)}
    elif (neexistuje záznam podřízeného volání) then
        založit záznam pro výpočet ψ(0) a přejít na něj
    elif (a nedefinováno) then
        a ← v_podřízené
        vymazat záznam podřízeného volání
        založit záznam pro výpočet ψ(1) a přejít na něj
    else
        b ← v_podřízené
        vymazat záznam podřízeného volání
        v ← a op(Q) b
        if (existuje nadřízené volání) then
            přejít na nadřízené volání
        else
            return(v)
        fi
    fi
od
```

Obrázek 4: Vyhodnocení formule tvaru QBF rozepsané do cyklu.

Pro vyhodnocení konstantního výrazu můžeme použít simulaci zásobníku na pásce TS. Pro jednoduchost budeme předpokládat, že výraz je uzávorkován tak, že k jeho vyhodnocení stačí předpokládat asociativitu spojek a není nutné určovat precedenci spojek kromě negace, která má prioritu větší než binární spojky. Například výraz $(1 \wedge 0 \wedge \neg 1) \vee (\neg 0 \wedge 1) \vee \neg 1$ je přípustný podle uvedených požadavků. Algoritmus prochází vstupní výraz odleva po jednotlivých symbolech. Každý symbol je uložen na vrchol zásobníku a pak se testuje, zda posloupnost několika nejvyšších symbolů na zásobníku je levá strana některého z pravidel v Obrázku 5. Pokud dojde ke shodě, je nejvyšší část zásobníku tvořící levou stranu pravidla nahrazena pravou stranou. Toto nahrazování se opakuje, dokud existují pravidla, kterými lze nahradit nejvyšší část zásobníku. Po přečtení celého výrazu a provedení popsaného postupu je v zásobníku jeden znak určující hodnotu výrazu.

Vypočtěme odhad prostoru potřebného pro celý výpočet podle programu na Obrázku 4. Nechť s je délka vstupní formule. Při každém rekurzivním vo-

$$
\begin{aligned}
0 \wedge 0 &\rightarrow 0\\
0 \wedge 1 &\rightarrow 0\\
1 \wedge 0 &\rightarrow 0\\
1 \wedge 1 &\rightarrow 1\\
0 \vee 0 &\rightarrow 0\\
0 \vee 1 &\rightarrow 1\\
1 \vee 0 &\rightarrow 1\\
1 \vee 1 &\rightarrow 1\\
\neg 0 &\rightarrow 1\\
\neg 1 &\rightarrow 0\\
(0) &\rightarrow 0\\
(1) &\rightarrow 1
\end{aligned}
$$

Obrázek 5: Pravidla pro substituce na zásobníku při vyhodnocování konstantního výrazu.

lání se sníží počet proměnných v předávané formuli. Protože původní počet proměnných je nejvýše s, je hloubka rekurze také nejvýše s. Jednotlivá volání procedury mají ve své oblasti pásky uloženu formuli, která byla danému volání předána, a informaci o výsledcích podřízených volání. Proto každé volání zabere na pásce místo velikosti $s+O(1)$. Celkový prostor je tedy $O(s^2)$, což je polynom od délky vstupní formule. □

Pro převod libovolné PSPACE úlohy L na QBF budeme popisovat posloupnosti konfigurací TS pro L, které tvoří výpočet. Přitom bude podstatné znát horní odhad počtu možných konfigurací TS v prostoru ℓ, který je formulován v následujícím lemmatu pro obecný TS. Pro jazyk v PSPACE bude navíc možné zvolit ℓ rovné polynomu od délky vstupu.

Lemma 6.2.2 *Pro každý jednopáskový TS existují konstanty $c_1, c_2 \in \mathrm{N}$ tak, že existuje nejvýše $2^{c_1\ell+c_2}$ konfigurací v prostoru ℓ.*

Důkaz: Jestliže Γ je pracovní abeceda uvažovaného TS a Q jeho množina stavů, pak konfigurací v prostoru ℓ je nejvýše $\ell \cdot |\Gamma|^\ell \cdot |Q| \leq (2|\Gamma|)^\ell \cdot |Q| = 2^{\log_2(2|\Gamma|)\cdot\ell+\log_2|Q|}$. Tvrzení věty pak platí pro $c_1 = \lceil \log_2(2|\Gamma|) \rceil$ a $c_2 = \lceil \log_2 |Q| \rceil$. □

Věta 6.2.3 *Každý jazyk L ze třídy* PSPACE *je p-převoditelný na* QBF*, i když se omezíme jen na formule s vnitřní částí ve tvaru CNF nebo jen ve tvaru DNF.*

Důkaz: Nechť $L \in$ PSPACE a nechť M rozpoznává L v prostoru $p(n)$, kde $p(n)$ je polynom. Popíšeme funkci f vyčíslitelnou v polynomiálním čase, která každému slovu w přiřadí nějakou uzavřenou kvantifikovanou formuli $f(w)$. Navíc,

$f(w)$ bude pravdivá právě tehdy, když $w \in L$. Ke konstrukci použijeme zápis konfigurace TS M pomocí vektoru Booleovských hodnot podobně jako v důkazu NP-úplnosti problému CSAT. Při značení podle Definice 2.5.2 má vektor pro konfiguraci v prostoru ℓ tvar $x_1, \ldots, x_\ell, y_1, \ldots, y_\ell$, kde x_i jsou kódy symbolů na pásce v nějakém kódování $k_1 : \Gamma \to \{0,1\}^{d_1}$ a y_i je buď kód prázdného symbolu nebo stavu řídící jednotky TS v nějakém kódování $k_2 : Q \cup \{\varepsilon\} \to \{0,1\}^{d_2}$. Vektory popisující konfigurace budou označovány symboly α, β, γ.

Popíšeme konstrukci formulí, které vyjadřují vlastnosti konfigurací podle Tabulky 1. Ke konstrukci formule Konfig_ℓ budeme potřebovat formuli, která vyjadřuje, že z určité skupiny proměnných má právě jedna hodnotu 1. Tuto skutečnost vyjadřuje formule

$$\mu_n(z_1, z_2, \ldots, z_n) =_{def} (z_1 \vee z_2 \vee \ldots \vee z_n) \bigwedge_{i<j} (\neg z_i \vee \neg z_j).$$

Kromě toho použijeme formule uvedené v Tabulce 2. S využitím těchto formulí zkonstruujeme Konfig_ℓ jako

$$
\begin{aligned}
\mathrm{Konfig}_\ell(\alpha) = {} & \mathrm{symbol}(x_1) \wedge \ldots \wedge \mathrm{symbol}(x_\ell)\\
& \wedge (\mathrm{stav}(y_1) \vee \mathrm{prazdny}(y_1)) \wedge \ldots \wedge (\mathrm{stav}(y_\ell) \vee \mathrm{prazdny}(y_\ell))\\
& \wedge \mu_\ell(\mathrm{stav}(y_1), \ldots, \mathrm{stav}(y_\ell)).
\end{aligned}
$$

Délka této formule je $O(\ell^2)$.

Formule	Význam
$\mathrm{Konfig}_\ell(\alpha)$	vektor α je kódem konfigurace
$\mathrm{Rovnost}_\ell(\alpha, \beta)$	vektory α a β jsou totožné
$\mathrm{Krok}_\ell(\alpha, \beta)$	β vznikne z α nejvýše jedním krokem výpočtu
$\mathrm{Cesta}_{\ell,s}(\alpha, \beta)$	β vznikne z α výpočtem délky nejvýše 2^s
$\mathrm{Pocatecni}_{w,\ell}(\alpha)$	α je počáteční konfigurací pro slovo w
$\mathrm{Prijimajici}_\ell(\alpha)$	α je přijímající konfigurací s hlavou TS na první pozici pásky

Tabulka 1: Formule vyjadřující vlastnosti konfigurací

Formule	Význam
$\mathrm{symbol}(x)$	$x \in \{0,1\}^{d_1}$ je kódem znaku ze Σ
$\mathrm{symbol}(x, a)$	$x \in \{0,1\}^{d_1}$ je kódem znaku a
$\mathrm{prazdny}(y)$	$y \in \{0,1\}^{d_2}$ je kódem prázdného znaku
$\mathrm{stav}(y)$	$y \in \{0,1\}^{d_2}$ je kódem některého stavu z Q
$\mathrm{stav}(y, q)$	$y \in \{0,1\}^{d_2}$ je kódem stavu q

Tabulka 2: Formule rozlišující typy znaků

Formule $\mathrm{Rovnost}_\ell(\alpha, \beta)$ je konjunkcí $\ell(d_1 + d_2)$ ekvivalencí odpovídajících si proměnných ve vektorech α a β.

V důkazu Věty 2.5.3 byl popsán obvod, který odvodí kód jedné konfigurace TS z konfigurace předchozí a který se skládá z ℓ obvodů, z nichž každý má nejvýše $3(d_1 + d_2)$ vstupních bitů a $d_1 + d_2$ výstupních bitů. Formuli $\mathrm{Krok}_\ell(\alpha, \beta)$

získáme následovně. Obvod, který počítá kód následující konfigurace, převedeme na množinu formulí, z nichž každá vyjadřuje jeden bit kódu následující konfigurace. Každou z těchto formulí lze vyjádřit DNF velikosti nejvýše $(3d_1 + 3d_2)2^{3d_1+3d_2}$. Jako vstup pro tyto formule dosadíme odpovídající bity konfigurace α a jednotlivé bity výsledné konfigurace porovnáme s odpovídajícími bity konfigurace β. Protože d_1 a d_2 považujeme za konstanty, získáme formuli velikosti $O(\ell)$.

Formuli $\mathrm{Cesta}_{\ell,s}(\alpha,\beta)$ definujeme indukcí podle s. Pro $s = 0$ je

$$\mathrm{Cesta}_{\ell,0}(\alpha,\beta) = \mathrm{Rovnost}_\ell(\alpha,\beta) \vee \mathrm{Krok}_\ell(\alpha,\beta)$$

a pro $s \geq 1$ je

$$\begin{aligned}\mathrm{Cesta}_{\ell,s}(\alpha,\beta) = (\exists\gamma)[&\mathrm{Konfig}_\ell(\gamma) \wedge (\forall\delta_1)(\forall\delta_2)[\\ &((\mathrm{Rovnost}_\ell(\alpha,\delta_1) \wedge \mathrm{Rovnost}_\ell(\gamma,\delta_2))\\ &\vee (\mathrm{Rovnost}_\ell(\gamma,\delta_1) \wedge \mathrm{Rovnost}_\ell(\beta,\delta_2))) \Rightarrow \mathrm{Cesta}_{\ell,s-1}(\delta_1,\delta_2)]] \ .\end{aligned}$$

Ukážeme, že tato formule vyjadřuje, že konfigurace β vznikne z α po nejvýše 2^s krocích M. K tomu využijeme následující ekvivalentní úpravu formule $\mathrm{Cesta}_{\ell,s}(\alpha,\beta)$ na delší formuli, kterou je snadnější analyzovat.

Lemma 6.2.4

$$\mathrm{Cesta}_{\ell,s}(\alpha,\beta) \equiv (\exists\gamma)[\mathrm{Konfig}_\ell(\gamma) \wedge \mathrm{Cesta}_{\ell,s-1}(\alpha,\gamma) \wedge \mathrm{Cesta}_{\ell,s-1}(\gamma,\beta)].$$

Důkaz: Uvažme podformuli $\mathrm{Cesta}_{\ell,s}(\alpha,\beta)$ s universálními kvantifikátory pro δ_1,δ_2 a proberme možné kombinace hodnot δ_1,δ_2. Pokud $(\delta_1,\delta_2) = (\alpha,\gamma)$, je implikace ve formuli ekvivalentní $\mathrm{Cesta}_{\ell,s-1}(\alpha,\gamma)$. Pokud $(\delta_1,\delta_2) = (\gamma,\beta)$, je implikace ve formuli ekvivalentní $\mathrm{Cesta}_{\ell,s-1}(\gamma,\beta)$. Ve všech ostatních případech je implikace splněna, a proto je uvažovaná podformule ekvivalentní konjunkci uvedených dvou případů. □

Lemma 6.2.5 *Formule* $\mathrm{Cesta}_{\ell,s}(\alpha,\beta)$ *je ekvivalentní výroku, že konfigurace* β *vznikne z* α *po nejvýše* 2^s *krocích* M *v prostoru nejvýše* ℓ.

Důkaz: Pro $s = 0$ plyne tvrzení z definice $\mathrm{Cesta}_{\ell,0}(\alpha,\beta)$. Pro $s \geq 1$ dokážeme tvrzení následovně. Jestliže je formule $\mathrm{Cesta}_{\ell,s}(\alpha,\beta)$ pravdivá, pak podle Lemma 6.2.4 existuje konfigurace γ tak, že platí současně $\mathrm{Cesta}_{\ell,s-1}(\alpha,\gamma)$ a $\mathrm{Cesta}_{\ell,s-1}(\gamma,\beta)$. Podle indukčního předpokladu z toho plyne, že z α do γ a z γ do β vedou výpočty délky nejvýše 2^{s-1}, tedy celkově z α do β výpočet délky 2^s. Naopak, jestliže existuje výpočet z α do β délky nejvýše 2^s, pak v něm existuje konfigurace γ, která dělí uvažovaný výpočet na úseky délky nejvýše 2^{s-1}. Podle indukčního předpokladu jsou tedy splněny formule $\mathrm{Cesta}_{\ell,s-1}(\alpha,\gamma)$ a $\mathrm{Cesta}_{\ell,s-1}(\gamma,\beta)$. Podle Lemma 6.2.4 je tedy splněna i formule $\mathrm{Cesta}_{\ell,s}(\alpha,\beta)$. □

Pro dokončení důkazu p-převoditelnosti obecné PSPACE úlohy na QBF bude potřeba ukázat, že zkonstruovaná formule má délku polynomiální vůči délce vstupní instance. Formule $\mathrm{Cesta}_{\ell,s}$ bude nejdelší částí výsledné formule, proto dokažme následující odhad.

Lemma 6.2.6 *Délka formule* $\mathrm{Cesta}_{\ell,s}$ *je* $O(\ell^2 s)$.

Důkaz: Předpokládejme, že ℓ je zafixováno. Formule $\mathrm{Cesta}_{\ell,0}$ má délku $O(\ell^2)$. Formule $\mathrm{Cesta}_{\ell,s}$ obsahuje jeden výskyt formule $\mathrm{Cesta}_{\ell,s-1}$ a pak další části, jejichž celková délka je $O(\ell^2)$. Délka formule $\mathrm{Cesta}_{\ell,s}$ je tedy součet $s+1$ příspěvků, z nichž každý je $O(\ell^2)$. Celá formule tedy má délku $O(\ell^2 s)$. □

Jestliže počáteční stav TS M je q_0, přijímající stav je q^+ a stroj se po ukončení výpočtu vždy vrátí na nejlevější pozici pásky, pak pro libovolné slovo $w = a_1 \ldots a_n$ mají formule $\mathrm{Pocatecni}_{w,\ell}$ a $\mathrm{Prijimajici}_\ell$ tvar

$$\begin{aligned}\mathrm{Pocatecni}_{w,\ell}(\alpha) = \ &\mathrm{Konfig}_\ell(\alpha) \wedge \mathrm{symbol}(x_1,a_1) \wedge \ldots \wedge \mathrm{symbol}(x_n,a_n)\\ &\wedge \mathrm{symbol}(x_{n+1},\varepsilon) \wedge \ldots \wedge \mathrm{symbol}(x_\ell,\varepsilon)\\ &\wedge \mathrm{stav}(y_1,q_0),\\ \mathrm{Prijimajici}_\ell(\alpha) = \ &\mathrm{stav}(y_1,q^+).\end{aligned}$$

Nyní již můžeme zkonstruovat formuli $f(w)$. Podle předpokladu věty je prostor potřebný k výpočtu stroje M pro slovo w nejvýše $p(n)$. Nechť c_1, c_2 jsou konstanty z Lemmatu 6.2.2 a zvolme $\ell = p(|w|)$ a $s = c_1\ell + c_2$. Pokud je slovo w přijato, je přijato výpočtem v prostoru nejvýše ℓ. Lemma 6.2.2 implikuje, že počet všech konfigurací M v prostoru ℓ je nejvýše 2^s. V přijímajícím výpočtu se nemohou opakovat stejné konfigurace. Pokud je tedy slovo w přijato, je přijato výpočtem délky nejvýše 2^s.

Podle Lemmatu 6.2.5 je tedy slovo w přijato právě tehdy, když je pravdivá následující formule, kterou označme jako $f_0(w)$.

$$(\exists\alpha)(\exists\beta)[\mathrm{Pocatecni}_{w,\ell}(\alpha) \wedge \mathrm{Prijimajici}_\ell(\beta) \wedge \mathrm{Cesta}_{\ell,s}(\alpha,\beta)],$$

kde ℓ a s jsou výše určené hodnoty.

Převedením $f_0(w)$ na prenexní tvar získáme formuli $f_1(w)$. Délka $f_1(w)$ je stejná jako délka $f_0(w)$. Nechť $f_1(w)$ má tvar $(Q_1x_1)\ldots(Q_nx_n)\,\varphi(x_1,\ldots,x_n)$, kde Q_i jsou kvantifikátory a vnitřní část $\varphi(x_1,\ldots,x_n)$ je obecná výroková formule. Formuli $f_1(w)$ ještě upravíme na ekvivalentní formuli $f(w)$, jejíž vnitřní část je v CNF.

Důkaz NP-úplnosti SAT ve Větě 3.3.4 využívá konstrukci, která ke každému Booleovskému obvodu $C(x_1,\ldots,x_n)$ zkonstruuje v polynomiálním čase formuli $\theta(x_1,\ldots,x_n,y_1,\ldots,y_m)$ ve tvaru CNF, která pro každé ohodnocení proměnných $x_1,\ldots,x_n$ splňuje

$$C(x_1,\ldots,x_n) = (\exists y_1)\ldots(\exists y_m)\theta(x_1,\ldots,x_n,y_1,\ldots,y_m)\ . \qquad (11)$$

Podformuli $\varphi(x_1,\ldots,x_n)$ ve formuli $f_1(w)$ můžeme interpretovat jako obvod $C(x_1,\ldots,x_n)$ a použít pro něj uvedenou konstrukci. Ve formuli $f_1(w)$ pak nahradíme podformuli $\varphi(x_1,\ldots,x_n)$ odpovídající pravou stranou (11) a výslednou formuli označíme $f(w)$. Z konstrukce plyne, že $f(w)$ je v normálním tvaru a je ekvivalentní formuli $f_1(w)$.

Formuli $f(w)$ je možné sestrojit také tak, že její vnitřní část je ve tvaru DNF. K tomu stačí provést konstrukci z předchozího odstavce pro formuli $\neg\varphi(x_1,\ldots,x_n)$. Pak je $C(x_1,\ldots,x_n) = \neg\varphi(x_1,\ldots,x_n)$, a tedy $\varphi(x_1,\ldots,x_n)$ je

ekvivalentní $\neg C(x_1, \ldots, x_n)$ a tedy také negaci pravé strany (11), což je obecně kvantifikovaná DNF.

Konstrukci formule $f(w)$ lze provést v polynomiálním čase v závislosti na $|w|$ a navíc pro každé w platí $w \in L \iff f(w) \in$ QBF. Jazyk L je tedy p-převoditelný na QBF při kterémkoli ze zvolených omezení na tvar vnitřní části formule a Věta 6.2.3 je tedy dokázána. □

Shrnutím vět 6.2.1 a 6.2.3 dostaneme:

Důsledek 6.2.7 *Problém* QBF *je* PSPACE*-úplný, i když se omezíme jen na formule s vnitřní částí ve tvaru CNF nebo jen ve tvaru DNF.*

Poznamenejme, že formule v normálním tvaru, které obsahují obecnou kvantifikaci CNF nebo existenční kvantifikaci DNF, lze snadno zjednodušit. Jestliže x je proměnná a $c_1, \ldots, c_m$ jsou klauzule, pak $(\forall x)(c_1 \wedge \ldots \wedge c_m)$ je ekvivalentní formuli $c'_1 \wedge \ldots \wedge c'_m$, kde c'_i vznikne z c_i vypuštěním x nebo $\neg x$, pokud se některý z těchto literálů v c_i vyskytuje, a $c'_i = c_i$ v opačném případě. Duální postup lze použít pro existenčně kvantifikovanou DNF.

6.3 Příklad PSPACE-úplné hry

Abychom mohli mluvit o PSPACE úplnosti, popíšeme nekonečnou množinu her. PSPACE úplná úloha pak bude, pro libovolnou hru z uvažované množiny rozhodnout, zda má vítěznou strategii pro prvního hráče nebo nemá.

Každá z uvažovaných her je popsána orientovaným grafem, v němž je vyznačen jeden vrchol. Hra předpokládá, že na vyznačeném vrcholu je položen hrací kámen. Hráči se pravidelně střídají v tazích. Jestliže kámen je v uzlu u, pak hráč, který je na tahu, přesune kámen do některého uzlu, do kterého vede z u hrana a který ještě nebyl během hry navštíven. Prohrává ten hráč, který nemá platný tah, tj. všechny hrany z právě navštíveného uzlu vedou do uzlů již dříve navštívených. Počet kroků hry je omezen počtem vrcholů grafu.

Název: Konečná hra na grafu.
Vstup: Orientovaný graf s vyznačeným uzlem.
Výstup: Zjistit, zda ve hře pro zadaný graf má první hráč vítěznou strategii.

Věta 6.3.1 *Problém konečná hra na grafu je v* PSPACE.

Důkaz: Algoritmus pro vyhodnocení hry bude procházet strom všech možných průběhů hry, přičemž z každého uzlu bude možná pokračování vybírat v libovolném pořadí hran. Tento postup lze popsat jako rekurzivní funkci, která jako vstup dostává aktuální konfiguraci hry, což je graf, ve kterém jsou vyznačeny již navštívené uzly a uzel, který byl navštíven naposledy. Výstupem funkce je zjištění, zda existuje vítězná strategie pro hráče, který je právě na tahu. Pro každý možný tah v dané konfiguraci je volána rekurze pro konfiguraci, která tímto tahem vznikne, a označme jako $y_1, \ldots, y_k$ výsledky těchto podřízených volání, kde k je počet platných tahů v aktuální konfiguraci. Pokud je $k = 0$, je výsledek pro aktuální konfiguraci 0, protože hráč, který je na tahu, nemá platný tah. Pokud $k \geq 1$, je výsledek $\bigvee_{i=1}^{k} \neg y_i = \neg \bigwedge_{i=1}^{k} y_i$, protože hodnota

této formule je 1 právě tehdy, když hráč, který je na tahu, může hru převést do konfigurace, která není vítězná pro hráče, který bude na tahu v dalším kroku.

Počet uzlů grafu označme n. Graf se v průběhu hry nemění. K popisu konfigurace jsou postačující následující informace

- index naposledy navštíveného uzlu,
- ohodnocení každého uzlu grafu bitem, který určuje, zda byl uzel již navštíven.

K uložení techto informací stačí prostor $O(n)$. Protože hloubka rekurze je omezena počtem uzlů grafu, lze celý výpočet realizovat v prostoru $O(n^2)$.

Pokud navštíveným uzlům přiřazujeme číslo kroku, kdy byl uzel navštíven, není nutné vytvářet kopii vektoru ohodnocení uzlů grafu pro každé rekurzivní volání a prostor lze snížit na $O(n \log n)$. □

Věta 6.3.2 *Problém konečná hra na grafu je* PSPACE *úplný.*

Důkaz: Ukážeme redukci problému QBF pro formule s vnitřní částí ve tvaru CNF na problém konečná hra na grafu. Nechť φ je libovolná taková formule. Zkonstruujeme graf $f(\varphi)$ s vyznačeným vrcholem tak, že hra pro graf $f(\varphi)$ má vítěznou strategii pro prvního hráče právě tehdy, když φ je pravdivá.

Pro formule s vnitřní částí DNF je možné konstrukci provést tak, že se níže popsaná konstrukce provede pro negaci vstupní formule, což je formule s vnitřní částí CNF. Zkonstruovaná instance hry na grafech se pak upraví tak, že se vymění role prvního a druhého hráče. K tomu stačí přidat nový počáteční uzel a jednu hranu, která vede z nového počátečního uzlu do původního počátečního uzlu.

Konstrukce grafu vyžaduje, aby se kvantifikátory ve formuli pravidelně střídaly počínaje existenčním a aby jich byl lichý počet. Pokud tuto podmínku formule nesplňuje, postupujeme ve formuli odleva a kvantifikátory, které chybí ke splnění podmínky doplníme jako kvantifikátory na nové proměnné, které se ve vnitřku formule a v předchozích kvantifikátorech nevyskytují. Tím se počet kvantifikátorů ve formuli zvětší nejvýše z k na $2k + 1$. Nová formule je ekvivalentní formuli původní. Nechť má nová formule tvar

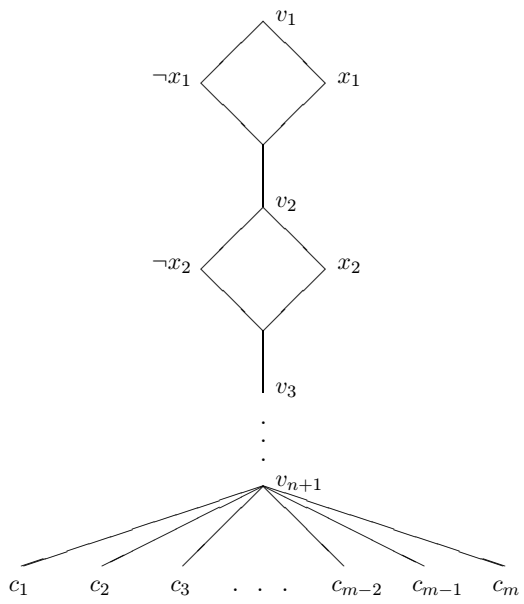
$$\varphi' \equiv (\exists x_1)(\forall x_2)(\exists x_3) \ldots (\exists x_n)\psi(x_1, \ldots, x_n) \equiv (Q_1 x_1) \ldots (Q_n x_n)\psi(x_1, \ldots, x_n),$$

kde $\psi(x_1, \ldots, x_n)$ je konjunkce klausulí $c_1, c_2, \ldots, c_m$.

Část grafu $f(\varphi)$ je na následujícím obrázku. Všechny hrany v obrázku jsou orientovány směrem dolů. Za vyznačený uzel považujeme uzel v_1.

Kromě hran uvedených v obrázku patří do grafu $f(\varphi)$ ještě všechny hrany, které spojují vrchol odpovídající některé klausuli s vrcholy, které odpovídají literálům obsaženým v dané klausuli. Tyto hrany jsou orientovány směrem od klausulí k literálům.

Hru si budeme představovat tak, že hráči postupně konstruují ohodnocení proměnných. Hráč, který vybírá hranu z vrcholu v_i určí svým výběrem hodnotu x_i tak, že tah do uzlu označeného x_i resp. $\neg x_i$ znamená ohodnocení proměnné



x_i hodnotou 0 resp. 1. Hodnota x_i je tedy určena tak, že je splněn ten z dvojice literálů x_i a $\neg x_i$, kterým cesta neprochází a může tedy být v dalším vývoji hry ještě použit. Protože mezi vrcholy v_i a v_{i+1} je lichý počet tahů, hráči se ve volbě proměnných pravidelně střídají. Posloupnost ohodnocení proměnných jednoznačně určuje cestu, kterou hra proběhla.

Pro důkaz souvislosti popsané hry a formule φ' zaveďme pro $i = 1, \ldots, n+1$ formule

$$S_i(x_1, \ldots, x_{i-1}) = (Q_i\, x_i) \ldots (\exists x_n)\ \psi(x_1, \ldots, x_n)$$

s volnými proměnnými $x_1, \ldots, x_{i-1}$. Speciálně, S_1 je ekvivalentní uzavřené formuli φ' a $S_{n+1}(x_1, \ldots, x_n)$ je ekvivalentní otevřené formuli $\psi(x_1, \ldots, x_n)$.

Tvrdíme, že pro libovolné $i = 1, \ldots, n+1$ a libovolné ohodnocení proměnných $x_1, \ldots, x_{i-1}$ je formule $S_i(x_1, \ldots, x_{i-1})$ pravdivá právě tehdy, když má první hráč vítěznou strategii v situaci, kdy se hra dostala do uzlu v_i po $i-1$ krocích odpovídajících ohodnocení proměnných $x_1, \ldots, x_{i-1}$. Tvrzení dokážeme indukcí pro $i = n+1, n, \ldots, 1$.

Nejprve dokažme, že formule S_{n+1} popisuje situaci v uzlu v_{n+1}. Uvažme situaci, kdy hra proběhla až do tohoto uzlu. Průběhem hry je určeno nějaké ohodnocení proměnných $x_1, \ldots, x_n$. Protože $n+1$ je sudé, je na tahu druhý hráč, který táhne do uzlu odpovídajícího některé klausuli. Z ní vedou hrany pouze do uzlů, které odpovídají literálům dané klausule. Pokud tah druhého hráče vede do klauzule, která je splněna, existuje v ní literál, který je splněn. Tento literál je dosud nenavštíven a první hráč může zvolit odpovídající uzel pro další tah a tím vyhrát, protože další tah již není možný. Pokud tah druhého

hráče vede do klauzule, která není splněna, není další tah možný a druhý hráč vyhrává. V uzlu v_{n+1} tedy má první hráč vítěznou strategii právě tehdy, když jsou všechny klauzule formule $\psi(x_1, \ldots, x_n)$ splněny, tedy když je tato formule splněna. Formule $S_{n+1} = \psi(x_1, \ldots, x_n)$ je tedy ekvivalentní existenci vítězné strategie pro prvního hráče v uzlu v_{n+1}.

Z konstrukce formulí S_i plyne, že pro $i \leq n$ platí

$$S_i(x_1, \ldots, x_{i-1}) = (Q_i\, x_i)\ S_{i+1}(x_1, \ldots, x_i)$$

Pro liché i je v uzlu v_i na tahu první hráč a má tedy vítěznou strategii právě tehdy, když alespoň jeden jeho tah vede do situace s vítěznou situací pro prvního hráče v uzlu v_{i+1}. Podle indukčního předpokladu je to ekvivalentní pravdivosti formule $(\exists x_i) S_{i+1}(x_1, \ldots, x_i)$. Protože Q_i je v tomto případě existenční kvantifikátor, je tato formule totožná s $S_i(x_1, \ldots, x_{i-1})$. Tím je pro liché i indukční krok dokázán. Pro sudé i je v uzlu v_i na tahu druhý hráč. První hráč má tedy v této situaci vítěznou strategii právě tehdy, když oba možné tahy druhého hráče vedou do situace s vítěznou strategií pro prvního hráče v uzlu v_{i+1}. Podle indukčního předpokladu je to ekvivalentní pravdivosti formule $(\forall x_i) S_{i+1}(x_1, \ldots, x_i)$. Protože Q_i je v tomto případě obecný kvantifikátor, je tato formule totožná s $S_i(x_1, \ldots, x_{i-1})$. Tím je pro sudé i indukční krok dokázán.

Podle dokázaného tvrzení je formule S_1 ekvivalentní existenci vítězné strategii na začátku hry. Protože S_1 je rovna formuli φ', je tím věta dokázána. □