## 1. Algoritmy a problémy

V matematice se často setkáváme s tvrzeními týkajícími se objektů, které neumíme zkonstruovat. Můžeme např. definovat množinu, na které dva homomorfismy (např. volné pologrupy) nabývají stejných hodnot, nebo množinu posloupností generujících prvků grupy, jejichž součin je roven jedné. Zájem o konstruktivní stránku matematiky byl v moderní matematice spíše okrajový. Rozvoj výpočetní techniky, který poskytl matematice dříve netušené možnosti, zájem o problematiku konstruktivního přístupu výrazně posílil. Možnosti poskytnuté počítačem obrátily pozornost k hranicím těchto možností. Navzdory naivnímu předpokladu, že hranice možností výpočetní techniky spočívají pouze v technické stránce věci, se objevila celá vědní disciplína týkající se samotné *možnosti* nějaký problém algoritmicky vyřešit. Ukázalo se totiž, že některé jednoduše formulované problémy takové algoritmické řešení nemají, a to nikoli z důvodu nedostatečného důvtipu programátorů nebo technické nedokonalosti hardwaru, nýbrž z hlubokých a podstatných teoretických důvodů. Zřejmě nejslavnějším příkladem takové *nerozhodnutelné* úlohy je tzv. Hilbertův desátý problém, který byl součástí seznamu předloženého slavným německým matematikem Davidem Hilbertem v roce 1900 jako výzva pro matematiku 20. století:

> 10. **Určení řešitelnosti diofantické rovnice**. *Je dána diofantická rovnice s libovolným počtem neznámých a racionálními číselnými koeficienty*: je třeba sestavit postup, kterým je možno konečným počtem kroků určit, zda je rovnice řešitelná v oboru celých čísel.
>
> (10. **Entscheidung der Lösbarkeit einer diophantischen Gleichung**. *Eine diophantische Gleichung mit irgendwelchen Unbekannten und mit ganzen rationalen Zahlkoefficienten sei vorgelegt*: man soll ein Verfahren angeben, nach welchen sich mittels einer endlichen Anzahl von Operationen entscheiden lässt, ob die Gleichung in ganzen rationalen Zahlen lösbar ist.)

*Negativní* řešení tohoto problému, tedy důkaz, že takový postup neexistuje, podal v roce 1970 Jurij Matijasevič.

Tvrzení, že neexistuje postup, který by v konečném počtu kroků vyřešil nějaký problém, vyžaduje upřesnění. Takovým postupem by totiž mohla být např. soustředěná snaha všech žijících matematiků o dané rovnici rozhodnout, zda je, nebo není řešitelná. Není jasné, jak by bylo vůbec možné dokázat, že takový postup nevede v konečném čase k cíli. Od postupu, požadovaného Hilbertem, očekáváme intuitivně několik vlastností:

- *Uniformita.* S každým případem se zachází stejně.
- *Jednoznačnost.* V každé fázi postupu musí být zřejmé, jak pokračovat. (Případně musí existovat konečně mnoho pokračování, mezi nimiž se předepsaným postupem, např. hodem mincí, vybírá.)
- *Mechaničnost.* Postup nesmí vyžadovat žádný důvtip. Musí být realizovatelný mechanickým zařízením, strojem, resp. důkladným úředníkem bez vlastního názoru.

V průběhu dvacátého století bylo navrženo několik modelů popisujících, jak má takový postup vypadat. Budeme je nazývat *výpočetní modely.* Nejslavnější a nejdůležitější výpočetní model navrhl Alan Turing ve svém článku z roku 1936

> On computable numbers with an application to the Entscheidungsproblem, Proc. London Math. Soc. 42 (1936) 230–265.

Po svém tvůrci se nazývá Turingův stroj (Turing machine) a budeme se jím podrobně zabývat. Zmíněný článek je pozoruhodný tím, že obsahuje nejen geniální intuice, ale také detailní rozpracování problému až k popisu tzv. univerzálního stroje, což je vlastně první „operační systém“.

Turing také důkladně vysvětluje, proč se domnívá, že jím navržený výpočetní model zahrnuje všechny postupy, které je možno rozumně považovat za algoritmus ve výše uvedeném smyslu. Toto přesvědčení se proslavilo pod názvem Churchova, resp. Churchova-Turingova teze:

> **Churchova-Turingova teze**: Každý výpočetní postup lze realizovat Turingovým strojem.

Churchova teze není matematické tvrzení, protože pojem výpočetního postupu je založen pouze na intuici. Soustředěný zájem všech matematiků např. za výpočetní postup nepovažujeme, protože nesplňuje (přinejmenším) podmínku mechaničnosti. Kde je ale přesná hranice výpočetních postupů není apriori jasné. Přesto je tato teze všeobecně přijímána, protože argumenty uváděné Turingem v její prospěch jsou velmi přesvědčivé a navíc ji potvrzuje intenzivní výzkum v průběhu celého dvacátého století, kdy se ukázalo, že všechny alternativní výpočetní modely jsou s Turingovým strojem ekvivalentní (nebo slabší).

Všeobecný souhlas s Churchovou tezí vede k tomu, že bývá obracena a používána jako *definice* výpočetního postupu:

**Definice 1.1** (Obrácená Churchova teze)**.** *Výpočetním postupem (algoritmem)* je to a jen to, co lze realizovat Turingovým strojem.

Protějškem algoritmu je *algoritmický problém*. Uveďme několik příkladů:

*Příklad* 1.2.
(1) Pro dané přirozené číslo najdi nějaký jeho faktor.
(2) Zjisti, zda dané přirozené číslo je prvočíslo.
(3) Zjisti, zda daná diofantická rovnice má řešení.
(4) Najdi nejkratší cestu mezi dvěma body orientovaného grafu.
(5) Zjisti, zda v daném grafu existuje hamiltonovská cesta.

Algoritmický problém se liší od obyčejného problému v tom, že nehledáme odpověď pro jednotlivý případ (např. jednotlivý graf), ale algoritmus, který najde odpověď pro všechny možné vstupy. Algoritmický problém je tedy množina *instancí*, tedy všech možných zadání problému. Tato množina je typicky nekonečná. Problémy s konečně mnoha instancemi nejsou z teoretického hlediska zajímavé, protože vždy existuje algoritmus, který spočívá v tabulce správných odpovědí. (Existují samozřejmě i problémy s konečně mnoha instancemi, které jsou z praktického hlediska zajímavé, např. problém rozhodnout v dané pozici na šachovnici, zda existuje vítězný tah.) Na druhou stranu, každá instance problému musí být konečná, aby mohla být zadána. Problém je tedy typicky *spočetná* množina instancí.

Podle typu odpovědi rozeznáváme problémy *vyhledávací* a *rozhodovací*. U rozhodovacího problému očekáváme odpověď ANO/NE (jsou to např. výše uvedené problémy 2, 3 a 5), vyhledávací problém má větší počet možných odpovědí (např. problémy 1 a 4).

Z praktických důvodů často omezíme naše zkoumání na problémy rozhodovací. Toto omezení není tak zásadní, jak by mohlo vypadat. Vyhledávací problémy lze převést na sérii rozhodovacích problémů typu: „je $i$-tý bit odpovědi 0 nebo 1?“. Pokud je např. takový rozhodovací problém polynomiální, je také výsledný vyhledávací algoritmus polynomiální.

## 2. Turingovy stroje

Stroj, který jako univerzální výpočetní model navrhl Turing, má podobu na obě strany nekonečné pásky rozdělené na políčka a opatřené *hlavou*, která se v daném okamžiku nachází na jednom políčku pásky.

Políčka mohou být prázdná nebo nést různé *symboly* vybrané z nějaké konečné množiny. Hlava může číst a přepisovat symbol na políčku, na kterém se nachází,

a pohybovat se po pásce vpravo nebo vlevo. Kromě toho obsahuje vnitřní paměť, která může nabývat konečného počtu *stavů*.

Na počátku výpočtu je páska prázdná až na konečnou souvislou posloupnost políček, která nese *vstup*. Hlava je na prvním políčku vstupu.

Chování stroje se řídí programem, což je množina instrukcí předepisujících, co má hlava v daném okamžiku v závislosti na svém stavu a čteném symbolu dělat. Algoritmus je skryt právě v programu, z matematického hlediska je Turingův stroj totožný se svým programem.

Matematicky tedy definujeme Turingův stroj následovně.

**Definice 2.1** (Turingův stroj)**.** Nechť $\Sigma$ a $Q$ jsou neprázdné konečné množiny. Množinu $\Sigma$ nazveme množinou *symbolů*, množinu $Q$ množinou *stavů*. Množina $\Sigma$ obsahuje symbol $\square$, který znamená prázdné políčko. Množina $Q$ obsahuje jeden význačný prvek $q_s$, který nazýváme *počáteční stav*. Nechť $\mathcal{M} = \{\leftarrow, \rightarrow, -\}$.

*Program* Turingova stroje je libovolná podmnožina $P \subseteq Q \times \Sigma \times Q \times \Sigma \times \mathcal{M}$. Libovolný prvek $(q, s, q', s', m) \in P$ se nazývá *instrukce*. *Turingův stroj* je čtveřice $T = (Q, \Sigma, q_s, P)$.

Množinu všech Turingových strojů značíme **TM**.

Množinu konečných posloupností symbolů z množiny $\Sigma$ (včetně prázdné posloupnosti) značíme $\Sigma^*$. Takové posloupnosti píšeme typicky bez čárek a nazýváme je *slova*, přičemž prvky $\Sigma$ nazýváme *písmena*. Množinu neprázdných slov značíme $\Sigma^+$. Turingův stroj $T$ spolu se svým vstupem $x \in \Sigma^*$ definuje *výpočet*. Na počátku výpočtu je hlava umístěna na prvním symbolu vstupu. Výpočet sestává z jednotlivých *kroků*. Každý krok spočívá v aplikaci jedné instrukce a je dán *okamžitým popisem* stroje v daném okamžiku. Okamžitý popis je dvojice $(q, s) \in Q \times \Sigma$, kde $q$ je momentální stav hlavy a $s$ je symbol na políčku, na kterém se hlava právě nachází, tzv. *čtený symbol*.

Následující krok je dán instrukcí $(q, s, q', s', m) \in P$ a má tři složky:

- čtený symbol se změní z $s$ na $s'$,
- stav hlavy se změní z $q$ an $q'$,
- hlava provede pohyb $m$.

Možné pohyby hlavy jsou tři. $\leftarrow$ znamená pohyb na sousední levé políčko, $\rightarrow$ pohyb na sousední pravé políčko a $-$ znamená, že hlava zůstane na stejném políčku, na kterém byla na začátku kroku.

Výpočet skončí, pokud se stroj dostane do okamžitého popisu, pro který neexistuje žádná instrukce (což typicky znamená, že dosáhl *koncového* stavu).

Po ukončení výpočtu zůstane na pásce *výstup*, což je posloupnost neprázdných symbolů, které se na pásce vyskytují.

Pokud výpočet neskončí, řekneme, že výpočet stroje $T$ na vstupu $x$ *diverguje*. Jinak říkáme, že *konverguje*.

Na Turingovy stroje se často kladou dodatečné požadavky, které usnadňují jejich studium. Takové dodatečné požadavky budeme vznášet, kdykoli to bude vhodné. Vždy by mělo být zřejmé, že nemají vliv na zkoumané vlastnosti. K nejčastějším požadavkům patří:

- Výpočet, pokud konverguje, končí vždy v jednom koncovém stavu, který se jinak během výpočtu nepoužívá.
- Stroj nikdy nezapisuje prázdný symbol. V tomto případě se obvykle zavádí nový pseudoprázdný symbol, který odliší prázdné nenavštívené políčko od prázdného navštíveného a který není považován za součást výstupu.
- Výstup je souvislý, není přerušen prázdnými ani pseudoprázdnými symboly.
- Hlava nikdy nezůstává na místě. Množina $\mathcal{M}$ se tedy omezuje na $\{\leftarrow, \rightarrow\}$

Naše definice Turingova stroje nevyžaduje, aby pro daný okamžitý popis existovala jediná instrukce. Pokud existuje instrukcí více, není výpočet definován jednoznačně. Turingovy stroje jsou tedy obecně *nedeterministické*. Pokud chceme tuto skutečnost zdůraznit, píšeme **NTM** místo **TM**.

Pokud má stroj $T$ pro každý okamžitý popis jedinou instrukci, nazývá se *deterministický*. Množinu deterministických Turingových strojů označujeme **DTM**. Deterministický stroj chápeme jako zvláštní případ stroje nedeterministického, je tedy $\mathbf{DTM} \subsetneq \mathbf{NTM}$.

Okamžitý popis Turingova stroje je lokální, omezuje se na situaci v místě hlavy. Celkový stav pásky v daném okamžiku nazýváme *úplný popis*, *konfigurace* nebo *snímek* stroje.

Snímek zapisujeme

$$\sigma = \square^\omega u_1\, q\, u_2, \square^\omega,$$

kde $u_1u_2$ je úsek pásky obsahující neprázdné symboly. Pokud přijmeme konvenci, že stroj nezapisuje prázdné symboly, je $u_1u_2 \in (\Sigma \setminus \{\square\})^*$. Hlava čte první symbol z $u_2$ a je ve stavu $q$. Počáteční a koncové $\square^\omega$ můžeme vynechávat.

Počáteční snímek stroje se vstupem $x$ je tedy $q_s x$. Turingův stroj $T$ definuje relaci $\delta_T$ mezi snímky. Píšeme

$$\sigma_1 \xrightarrow{\delta_T} \sigma_2,$$

pokud snímek $\sigma_2$ vznikne ze snímku $\sigma_1$ jedním krokem.

*Příklad* 2.2. Instrukce $(q, 1, q', 0, \rightarrow)$ převádí snímek $10q10$ na $100q'0$.

Tranzitivní obal relace $\delta_T$ značíme $\delta_T^*$. Zápis

$$\sigma_1 \xrightarrow{\delta_T^*} \sigma_2,$$

tedy znamená, že existuje (libovolně dlouhý) výpočet, který převádí snímek $\sigma_1$ na snímek $\sigma_2$.

V případě deterministického Turingova stroje existuje pro každý vstup právě jeden výstup nebo stroj diverguje. Pro $T \in \mathbf{DTM}$ označujeme výstup $T$ na vstupu $x$ jako $T(x)$, přičemž pro divergující výpočet definujeme nový symbol $\nearrow$ a píšeme $T(x) = \nearrow$.

Skutečnost, že Turingův stroj je totožný se svým programem, může být matoucí. Očekáváme, že stroj se bude podobat spíše počítači, který programy vykonává. Počítač je ovšem také program, a sice operační systém, který umí všechny ostatní programy realizovat. To vede k důležité myšlence, že Turingův stroj může být reprezentován nějakým kódem, který sám může být vstupem jiného Turingova stroje. Turingovy stroje sice obecně obsahují neomezeně mnoho symbolů a stavů, ale všechny je lze zakódovat konečnou abecedou. Zvolme nějakou konečnou abecedu $\Sigma$ a fixujme nějaké kódování množiny **TM** Turingových strojů, které budeme dále nazývat *standardní*. Můžeme zvolit $\Sigma = \{0, 1\}$ (což se, jak dobře víme, skutečně děje ve výpočetní technice) nebo, pro lepší představu, nějakou bohatší abecedu, např. $\Sigma = \{0, 1, (, |, )\}$. Zakódovaný Turingův stroj pak může vypadat např. takto:

$$(0|0|1|0|1)(0|1|0|1|1)(1|1|1|1|1)(1|10|10|10|1)(0|10|10|0|1)$$

Kód Turingova stroje $T$ budeme značit $[T]$. Není nutné, abychom zde standardní kódování přesně definovali, protože ho nebudeme konkrétně používat. Důležitá ovšem je, aby toto kódování bylo „srozumitelné“ v tom smyslu, že existuje nějaký Turingův stroj $U$, který, když dostane na vstupu dvojici $([T], x)$, spočte $T(x)$ (včetně toho, že diverguje, pokud $T(x)$ diverguje). Takovému $U$ se říká *univerzální Turingův stroj*. Práce univerzálního stroje $U$ je založena na simulaci práce stroje $T$. Myšlenka, že všechny možné algoritmy je možné realizovat jediným strojem pomocí programu, který je uložen v paměti podobně jako vstup (tzv. *stored-program computer*), je

zásadním přínosem Alana Turinga a znamenala faktický začátek teoretické informatiky. Původní Turingův článek obsahuje detailní popis programu univerzálního stroje, což lze považovat za důkaz existence takového stroje. Možnost simulace stroje na základě rozumného zápisu jeho instrukcí je ovšem poměrně intuitivní a v dnešní době především důvěrně známý fakt.

## 3. Rekursivní funkce a jazyky

Vstupem i výstupem Turingova stroje je nějaký prvek $\Sigma^*$. Uvažujeme-li standardní kódování Turingových strojů, můžeme dokonce předpokládat, že $\Sigma$ je fixní pro všechny Turingovy stroje a stejné jako abeceda standardního kódování Turingových strojů.

Deterministický stroj $T$ tedy můžeme chápat jako zobrazení $\Sigma^* \to \Sigma^*$. (Namísto „zobrazení“ budeme také často říkat „funkce“.) To je v souladu se zvoleným zápisem $T(x) = y$, pokud je $y$ výstupem stroje $T$ na vstupu $x$. Zdůrazněme, že tato definice dává dobrý smysl pouze pro *deterministické* stroje. Pokud stroj na vstupu $x$ diverguje, píšeme $T(x) = \nearrow$, což je nejlepší chápat tak, že zobrazení $T$ není na $x$ definováno. Je to tedy zobrazení *částečné*.

$\Sigma^*$ je spočetná množina, lze ji tedy očíslovat přirozenými čísly. *Očíslováním* nějaké množiny $S$ rozumíme libovolnou bijekci $f : \mathbb{N} \to S$. Nejvhodnější očíslování množiny $\Sigma^*$ je dáno jeho maximo-lexikografickým uspořádáním $\prec$.

**Definice 3.1** (Maximo-lexikografické uspořádání)**.** Uvažujme nějaké lineární uspořádání prvků $\Sigma$ a příslušné lexikografické uspořádání $<$ na $\Sigma^*$.

*Maximo-lexikografické uspořádání* $\prec$ odpovídající $<$ je definováno pravidly:

a) Je-li $u$ delší než $v$, pak $v \prec u$.
b) Mají-li $u$ a $v$ stejnou délku, je $u \prec v$ právě když je $u < v$.

Takto definujeme bijekci $\nu : \mathbb{N} \to \Sigma^*$. $\nu(0)$ je prázdný řetězec a $\nu(i)$ je $i$-té neprázdné slovo v uspořádání $\prec$. Přirozenost tohoto uspořádání spočívá v tom, že pokud $\Sigma = \{1, 2, \ldots, k\}$ a $<$ je běžné uspořádání, pak $\nu(m)$ je $k$-adický zápis čísla $m$, tj.

$$c_j c_{j-1} \ldots c_1 = \nu\left(\sum_{i=1}^{j} c_i \cdot k^{j-i}\right).$$

O objektech, které lze algoritmicky zkonstruovat říkáme, že existují *efektivně*. Je zřejmé, že zobrazení $\nu$ efektivní je, tj. existuje Turingův stroj $T_\nu$ který na vstupu $n$ vrátí $\nu(n)$. Viděli jsme, že toto zobrazení je ve skutečnosti identita, používáme-li $k$-adický zápis čísla. Stejně tak je efektivní inverzní zobrazení $\nu^{-1}$.

Při daném očíslování $\Sigma^*$ přirozenými čísly, odpovídá každý deterministický Turingův stroj $T \in \mathbf{DTM}$ částečné funkci $f_T : \mathbb{N} \to \mathbb{N}$.

**Definice 3.2** (Rekursivní funkce)**.** Částečná funkce $f : \mathbb{N} \to \mathbb{N}$, pro kterou existuje nějaký $T \in \mathbf{DTM}$ takový, že $f = f_T$, se nazývá *částečná rekursivní*. Je-li částečná rekursivní funkce definovaná na celém $\mathbb{N}$, nazývá se *úplná rekursivní* nebo prostě *rekursivní*.

Množinu všech částečných rekursivních funkcí budeme značit **PRF** (partial recursive functions) a množinu rekursivních funkcí **RF**.

Řekli jsme, že každý deterministický Turingův stroj představuje nějakou částečnou rekursivní funkci. To ovšem neznamená, že můžeme množiny **DTM** a **PRF** ztotožnit. **DTM** obsahuje všechny deterministické Turingovy stroje, tedy všechny možné deterministické seznamy instrukcí. Mnoho různých Turingových strojů však počítá stejnou funkci. Stačí uvážit stroj, ke kterému přidáme stav, který nebude nikdy dosažen. Pro takový stav pak můžeme libovolně tvořit instrukce, aniž se příslušná funkce změní.

Právě vztah mezi (částečnými) funkcemi $f : \mathbb{N} \to \mathbb{N}$ a Turingovými stroji je předmětem teorie, kterou zde rozvíjíme. Každá funkce je nekonečný objekt (algoritmický problém s nekonečně mnoha instancemi), zatímco příslušný Turingův stroj je konečný reprezentant tohoto nekonečného objektu, resp. konečný postup umožňující tuto funkci počítat. Turingův stroj $T$ je přitom reprezentován svým (standardním) kódem $[T]$, což je také prvek $\Sigma^*$. Ne každý prvek $\Sigma^*$ je ovšem kódem nějakého Turingova stroje. Přesto platí následující tvrzení.

**Tvrzení 3.3.** *Standardní kódy (deterministických) Turingových strojů lze efektivně očíslovat.*

*Důkaz.* Kódy budeme číslovat podle maximo-lexikografického uspořádání $\prec$. Označme takové očíslování $G : \mathbb{N} \to \mathbf{TM}$. Hodnota $G(i)$ je kód Turingova stroje, který je $i$-tým slovem z **TM** vzhledem k maximo-lexikografickému uspořádání.

Ukažme, že funkce $G$ je rekursivní. Turingův stroj, který ji počítá, bude postupně v maximo-lexikografickém uspořádání generovat všechna slova ze $\Sigma^*$. U každého slova ověří, zda se jedná o platný kód (deterministického) Turingova stroje, a pokud ano, zvýší své počítadlo (vyhrazený kus pásky) o jedna. Ve chvíli, kdy na počítadle dosáhne vstupní hodnotu $i$, oznámí jako výstup poslední nalezený kód Turingova stroje. □

Myšlenka číslování množiny konečných posloupností pochází od Kurta Gödela, který ji poprve použil v kontextu formální logiky pro očíslování logických formulí. Na jeho počest používáme značení $G$. Takové očíslování se také někdy nazývá Gödelovo.

Částečnou rekursivní funkci počítanou strojem $G(i)$ budeme také značit $\varphi_i$. Posloupnost $(\varphi_i)_{i\in\mathbb{N}}$ je tedy výčtem všech **PRF**. Znovu ovšem připomeňme, že každá funkce se v seznamu vyskytuje vícekrát (dokonce nekonečně mnohokrát). Nakonec můžeme definovat *univerzální rekursivní funkci* $\Phi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ předpisem $\Phi(i, n) = \varphi_i(n)$. Je snadné si rozmyslet, že $\Phi$ je sama rekursivní (s přirozeným rozšířením naší definice rekursivních funkcí na funkce dvou – a více – proměnných, případně s použitím nějakého efektivního očíslování dvojic čísel).

Po předchozím positivním výsledku obrátíme svou pozornost k výsledkům negativním. Optimistická představa, podle které lze každý problém řešit algoritmicky, je neudržitelná už proto, že všech funkcí je nespočetně mnoho (kontinuum), zatímco Turingových strojů, a tedy rekursivních funkcí jen spočetně. Můžeme tedy dokonce říci, že funkce jsou v drtivé většině nerekursivní.

Problémy, které nás v matematice zajímají, jsou ovšem vždy formulovány nějakým konečným popisem. Není tedy předem zřejmé, zda existují nějaké nerekursivní přirozené problémy. Jak jsme uvedli v úvodu, ukazuje se, že i velmi konkrétní matematické problémy jsou nerekursivní, neboli *nerozhodnutelné*.

Všimněte si, že terminologie je z historických důvodů dosti rozrůzněná. Termíny *efektivní*, *rekursivní*, *rozhodnutelný* mají v podstatě totožný význam, ačkoli se používají v různých kontextech (efektivní číslování, rekursivní funkce, rozhodnutelný problém).

Historicky nejstarší a teoreticky nejdůležitější nerozhodnutelné problémy se týkají otázky, zda daný stroj na daném vstupu konverguje, což souvisí s otázkou zda je funkce počítaná daným stroj úplná. Označme **UTM** množinu deterministických strojů, které počítají úplnou funkci.

**Tvrzení 3.4.** *Neexistuje efektivní očíslování množiny* **UTM**.

*Důkaz.* Budeme postupovat sporem. Předpokládejme, že existuje rekursivní funkce $H : \mathbb{N} \to \mathbf{UTM}$, která je na. Uvažme funkci $F : \mathbb{N} \to \mathbb{N}$ definovanou předpisem

$$F(i) := f_{H(i)}(i) + 1.$$

Ukažme, že funkce $F$ je rekursivní. Díky rekursivitě $H$ lze pro dané $i$ efektivně nalézt stroj $M = H(i) \in \mathbf{UTM}$, který počítá funkci $f_{H(i)}$, a pomocí tohoto stroje pak spočítáme hodnotu $F(i)$ jako výstup $M$ na vstupu $i$ zvětšený o jedna.

Funkce $F$ je navíc úplná, a proto existuje stroj $T \in \mathbf{UTM}$, který ji počítá, a přirozené číslo $i_F$, pro které $H(i_F) = T$.

Spor dosáhneme otázkou po hodnotě funkce $F$ v bodě $i_F$. Podle definice $F$ platí

$$F(i_F) = f_{H(i_F)}(i_F) + 1$$

Avšak funkce

$$f_{H(i_F)}$$

je rovna $F$, a tudíž

$$F(i_F) = F(i_F) + 1,$$

spor. □

Hlavní myšlenka předchozího důkazu se nazývá diagonální argument, protože funkce $F$ byla definována jako diagonála tabulky hodnot funkcí $G(i)$ zvětšená o jednu. Stejný postup se používá v důkazu, že podmnožiny přirozených čísel nelze očíslovat přirozenými čísly.

Následující věta je prototyp a zdroj všech vět o nerozhodnutelnosti.

**Věta 3.5.** *Nelze rozhodnout, zda se Turingův stroj na daném vstupu zastaví (neboli zda je daná částečně rekursivní funkce na dané hodnotě definovaná).*

*Důkaz.* Nechť $G$ je efektivní číslování **DTM**, zaručené Tvrzením 3.3. Chceme použít diagonální argument a funkci $F$ podobně jako v předchozí větě. Nyní ovšem $G(i)$ může v bodě $i$ divergovat.

Definujme tedy $F$ takto

$$F(i) := \begin{cases} f_{G(i)}(i) + 1, & \text{pokud se } G(i) \text{ na vstupu } i \text{ zastaví,} \\ 0, & \text{pokud } G(i) \text{ na vstupu } i \text{ diverguje.} \end{cases}$$

Je-li funkce $F$ rekursivní, dostaneme spor jako v předchozí větě. Jediným důvodem, proč $F$ není rekursivní, může být neexistence algoritmu, který rozhodne, kterou ze dvou větví definice $F$ použít. Tím je věta dokázána. □

Předchozí věta se často formuluje slovy: HaltingProblem je nerozhodnutelný. Zdůrazněme, že nerozhodnutelnost problému neznamená, že pro něj nelze v konkrétním případě nalézt řešení. Znamená neexistenci algoritmu, který bude úspěšný pro *všechny* instance.

V předchozím textu jsme uvedli do souvislosti algoritmy a funkce $\mathbb{N} \to \mathbb{N}$. V případě rozhodovacích problémů, tj. takových, u kterých je výstupem odpověď ANO nebo NE, je situace ještě o něco jednodušší. Za obor hodnot příslušných funkcí můžeme považovat množinu $\{0, 1\}$ a funkce je charakteristickou funkcí množiny instancí s kladnou odpovědí.

Algoritmus má pro daný vstup rozhodnout, zda v množině leží. Vstupem jsou slova ze $\Sigma^*$, libovolná množina slov se nazývá *jazyk*.

Vzhledem k existenci efektivního očíslování $\Sigma^*$, odpovídá každý jazyk podmnožině $\mathbb{N}$. Máme tedy tři vzájemně si odpovídající úhly pohledu:

rozhodovací problémy $\equiv$ jazyky $\equiv$ množiny přirozených čísel

Pro *deterministické* stroje máme následující definice.

**Definice 3.6.** Řekneme, že stroj $T \in \mathbf{DTM}$ *přijímá* jazyk $L \subset \Sigma^*$, pokud $x \in L$, právě když $T(x) = 1$.

Řekneme, že stroj $T \in \mathbf{DTM}$ *rozhoduje* jazyk $L \subset \Sigma^*$, pokud

pro $x \in L$ platí $T(x) = 1$ a

pro $x \notin L$ platí $T(x) = 0$.

Zásadní rozdíl mezi přijímáním a rozhodováním jazyka spočívá v tom, že přijímající stroj může na vstupu $x \notin L$ divergovat (může také dávat jinou hodnotu než 0 nebo 1, ale to lze snadno změnit na odmítnutí vstupu).

Jazyk, který je přijímán nějakým deterministickým Turingovým strojem se nazývá *rekursivně spočetný*. Jazyk, který je nějakým takovým strojem rozhodován se nazývá *rekursivní*. Množinu rekursivně spočetných jazyků značíme **RE** (recursively enumerable) a množinu rekursivních jazyků **R**.

*Komplement* jazyka $L$ je jazyk $\overline{L} = \Sigma^* \setminus L$. Množinu jazyků, jejichž komplement je rekursivně spočetný značíme **co-RE**.

**Tvrzení 3.7.**

$$\mathbf{R} = \mathbf{RE} \bigcap \mathbf{co\text{-}RE}.$$

*Důkaz.* Je-li jazyk rekursivní pak je on i jeho komplement zřejmě rekursivně spočetný.

Naopak, jsou-li $L$ a $\bar{L}$ rekurzivně spočetné, pak existují Turingovy stroje $M_1$ a $M_2$ takové, že $M_1$ přijímá $L$ a $M_2$ přijímá $\bar{L}$. Zkonstruujeme Turingův stroj $T$ pomocí $M_1$ a $M_2$ následovně. Na vstupu $x$ bude $T$ souběžně simulovat chod obou strojů. Provede vždy střídavě jednu instrukci stroje $M_1$ se vstupem $x$ a jednu instrukci stroje $M_2$ se vstupem $x$. Jeden ze simulovaných strojů se po konečném počtu kroků zastaví a přijme svůj vstup. Stroj $T$ podle toho, který stroj vstup přijal, odpoví 1 nebo 0. □

Následující tvrzení poskytuje motivaci pro název „recursively enumerable“.

**Tvrzení 3.8.** *Pro jazyk $L$ jsou následující podmínky ekvivalentní:*

- *$L$ je rekursivně spočetný,*
- *$L$ je oborem hodnot nějaké částečné rekursivní funkce,*
- *$L$ je oborem hodnot nějaké úplné rekursivní funkce,*

*Důkaz.* Nechť je jazyk $L \subset \Sigma^*$ rekursivně spočetný a $S \in \mathbf{DTM}$ ho přijímá. Nechť

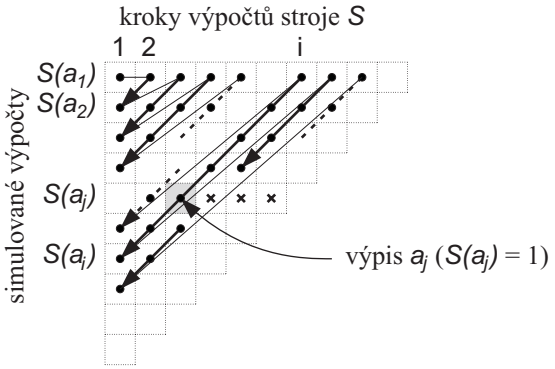
$$\Sigma^* = \{a_1, a_2, \dots\}$$

je nějaké efektivní očíslování. Zkonstruujeme stroj $M$ jehož oborem hodnot bude $L$. Běh stroje je znázorněn na Obrázku 1.

Rozdělme výpočet stroje $M$ na jednotlivá kola. V prvním kole bude stroje $M$ simulovat první krok stroje $S$ se vstupem $a_1$. V druhém kole stroje se bude simulovat druhý krok výpočtu $S(a_1)$ a první krok výpočtu $S(a_2)$. V $i$-tém kole se tedy bude počítat $i$-tý krok výpočtu $S(a_1)$, $(i-1)$-ní krok $S(a_2)$ atd. až po první krok výpočtu $S(a_i)$. Na obrázku 1 jsou jednotlivá kola stroje $M$ vyznačena jako tučné diagonální šipky. V $k$-tém řádku ($k \geq 1$) jsou znázorněny kroky stroje $S$ se vstupem $a_k$. Simulace jednoho kroku výpočtu stroje $S$ je vyznačena kroužkem. Pokud nastane $S(a_j) = 1$ pro nějaké $j \geq 1$ v $i$-tém kole výpočtu stroje $M$, pak se slovo $a_j$ vypíše a v následujících kolech stroje $M$ se již výpočet $S(a_j)$ nesimuluje (na obrázku označeno křížkem).

Protože stroj $M$ prochází všechna slova nad abecedou $\Sigma$ a platí inkluze $L \subseteq \Sigma^*$, musí být každé $x \in L$ po konečném počtu kroků stroje $M$ nalezeno. Nyní již snadno definujeme výstup stroje $M$ na vstupu $i \in \mathbb{N}$. $M$ bude počítat tak dlouho, až najde $i$ slov z jazyka $L$. Poslední z nich bude jeho výstupem. Je zřejmé, že $M$ počítá úplnou rekursivní funkci. Ta je současně triviálně i částečnou rekursivní funkcí.

Zbývá ukázat, že existuje-li Turingův stroj $S$, jehož obor hodnot je $L$, pak lze zkonstruovat stroj $M$, který jazyk $L$ přijímá. Při dotazu $x \in L$ začne $M$ získávat diagonální metodou (analogicky jako v důkazu první části) všechna slova jazyka $L$ od stroje $S$. Tato slova bude porovnávat se slovem $x$. Platí-li $x \in L$, pak se stroj $M$




Obrázek 1. Schéma výpočtu Turingova stroje $M$.

po konečném počtu kroků zastaví a odpoví správně. Stroj $M$ tedy přijímá jazyk $L$ a jazyk $L$ je rekurzivně spočetný. □

## 4. Nedeterministické stroje

Na Turingovy stroje se díváme jako na nástroje, umožňující efektivně vyřešit nějaký problém. Takový pohled je ale oprávněný jen pro deterministické stroje. Pro nedeterministické stroje nejen není jasné, jak je prakticky realizovat, není dokonce jasné ani to, co považovat za jejich výstup. Na daném vstupu totiž může existovat mnoho různých výpočtů, tedy také mnoho výsledků, relace $\delta_T^*$ nedefinuje funkci a symbol $T(x)$ nedává dobrý smysl.

Jak tedy chápat nedeterministické výpočty? Odpověď dává následující zásadní definice.

**Definice 4.1.** Řekneme, že nedeterministický stroj $A$ *přijímá vstup* $x$, právě když na vstupu $x$ existuje *alespoň jeden* přijímající výpočet.

Řekneme, že nedeterministický stroj $A$ *přijímá* jazyk $L$, pokud $A$ přijímá vstup $x$, právě když $x \in L$.

Je důležité si uvědomit, že definice přijímání vstupu nedeterministickým strojem je nekonstruktivní, mluví o existenci přijímajícího výpočtu, neříká ale nic o tom, jak takový výpočet najít, nebo jak s pomocí daného nedeterministického stroje vůbec zjistit, zda existuje. Jedna z možností, z myšlenkového hlediska nejjednodušší, ale časově náročná, je projít všechny možné výpočty; o tom mluví Věta 7.4.

Fungování nedeterministického stroje si lze ilustrovat na Tvrzení 3.8. Je-li dán deterministický stroj $M$ s oborem hodnot $L$, můžeme definovat nedeterministický algoritmus přijímající $L$ takto:

- vstup $x$;
- nedeterministicky zvol $i$;
- pokud $M(i) = x$, přijmi.

Postupné prohledávání všech výpočtů použité v důkazu Tvrzení 3.8. je vlastně deterministickou simulací uvedeného nedeterministického algoritmu.

V případě nedeterminismu se omezujeme na rozhodovací problémy. Proto se někdy říká, že nedeterministický stroj svůj jazyk *rozhoduje*. Jak je ale vidět i na předchozím příkladu, termín „přijímá“ lépe vyjadřuje asymetrii mezi přijetím a zamítnutím vstupu. Nedeterministický stroj může na vstupu z přijímaného jazyka umožňovat mnoho výpočtů, které nekončí přijetím; to ale neznamená zamítnutí. Vstup je „zamítnut“, jen pokud neexistuje *žádný* přijímající výpočet.

## 5. Složitostní třídy

Dosud jsme se zabývali jen otázkou, zda je daný problém vůbec algoritmicky řešitelný. Naším hlavním tématem však bude otázka, jak *složité* je řešení daného řešitelného problému.

Nejprve definujeme třídy funkcí, jimiž budeme složitost měřit.

**Definice 5.1.** Řekneme, že funkce $f$ leží v $\mathcal{O}(g)$, pokud existují $c > 0$ a $n_0 > 0$ takové, že pro všechna $n > n_0$ platí

$$|f(n)| < c \cdot |g(n)|.$$

Řekneme, že funkce $f$ leží v $o(g)$, pokud

$$\lim_{n\to\infty} \frac{|f(n)|}{|g(n)|} = 0.$$

Řekneme, že funkce $f$ leží v $\theta(g)$, pokud $f \in \mathcal{O}(g)$ a současně $g \in \mathcal{O}(f)$.
Řekneme, že funkce $f$ leží v $\Omega(g)$, pokud $g \in \mathcal{O}(f)$.
Řekneme, že funkce $f$ leží v $\omega(g)$, pokud $g \in o(f)$.

V literatuře se často namísto $f \in \mathcal{O}(g)$ píše $f = \mathcal{O}(g)$ (a podobně pro ostatní symboly).

Hlavními měřítky náročnosti daného výpočtu je spotřeba času a prostoru. Náročnost je vždy *funkcí délky vstupu*.

**Definice 5.2.** Nechť $A \in \mathbf{TM}$. Označme $t'_A(x)$ maximální počet kroků výpočtu stroje $A$ na vstupu $x \in \Sigma^*$ (o maximálním počtu kroků hovoříme proto, že stroj je obecně nedeterministický). *Časová náročnost* algoritmu $A$ je funkce $t_A : \mathbb{N} \to \mathbb{N}$, kde

$$t_A(n) = \max_{|x|=n}\{t'_A(x)\}.$$

Podobně označme $s'_A(x)$ maximální počet navštívených políček během výpočtu na vstupu $x$. *Prostorová náročnost* algortimu je funkce

$$s_A(n) = \max_{|x|=n}\{s'_A(x)\}.$$

K prostorové náročnosti viz Definici 6.2.

**Definice 5.3.** Řekneme, že problém $P$ leží v $\mathbf{TIME}(f)$ (resp. $\mathbf{NTIME}(f)$), pokud existuje algoritmus $A \in \mathbf{DTM}$ (resp. $\in \mathbf{NTM}$), který ho řeší, a platí $t_A \leq f$.

Je-li $F$ třída funkcí, píšeme $P \in \mathbf{TIME}(F)$, pokud $P \in \mathbf{TIME}(f)$, pro nějakou funkci $f \in F$.

Analogicky definujeme třídu problémů $\mathbf{NTIME}(F)$, $\mathbf{SPACE}(f)$, $\mathbf{NSPACE}(f)$, $\mathbf{SPACE}(F)$ případně $\mathbf{NSPACE}(F)$.

Protože jsme $t'_A(x)$ v Definici 5.2 určili jako *maximální* z možných výpočtů, musejí v nedeterministickém případě v daném čase skončit všechny možné výpočty, bez ohledu na jejich výsledek (např. polynomiální nedeterministický stroj se tedy v polynomiálním čase zastaví *vždy*).

O algoritmu, který ve výše uvedených definicích musí existovat, aby problém, který řeší, ležel ve třídě $\mathcal{C}$, řekneme, že třídě $\mathcal{C}$ *odpovídá*. Přestože jsou předchozí definice formulovány pro obecné problémy, budeme je používat především pro problémy rozhodovací. Pokud řekneme, že nějaký jazyk leží v třídě $\mathcal{C}$, znamená to, že je rozhodován nějakým $A \in \mathbf{TM}$, který odpovídá $\mathcal{C}$.

Některé důležité složitostní třídy mají zvláštní jména.

**Definice 5.4.**

$$\mathbf{P} = \bigcup_{k\geq 1} \mathbf{TIME}\left(n^k\right) = \mathbf{TIME}\left(n^{\mathcal{O}(1)}\right)$$

$$\begin{aligned}
\mathbf{NP} &= \bigcup_{k\geq 1} \mathbf{NTIME}\left(n^k\right) = \mathbf{TIME}\left(n^{\mathcal{O}(1)}\right)\\
\mathbf{EXP} &= \bigcup_{k\geq 1} \mathbf{TIME}\left(2^{n^k}\right) = \mathbf{TIME}\left(2^{n^{\mathcal{O}(1)}}\right)\\
\mathbf{E} &= \bigcup_{k\geq 1} \mathbf{TIME}\left(2^{kn}\right) = \mathbf{TIME}\left(2^{\mathcal{O}(n)}\right)\\
\mathbf{NEXP} &= \bigcup_{k\geq 1} \mathbf{NTIME}\left(2^{n^k}\right) = \mathbf{NTIME}\left(2^{n^{\mathcal{O}(1)}}\right)\\
\mathbf{L} &= \mathbf{SPACE}\left(\log(n)\right)\\
\mathbf{NL} &= \mathbf{NSPACE}\left(\log(n)\right)\\
\mathbf{PSPACE} &= \bigcup_{k\geq 1} \mathbf{SPACE}\left(n^k\right) = \mathbf{SPACE}\left(n^{\mathcal{O}(1)}\right)\\
\mathbf{NPSPACE} &= \bigcup_{k\geq 1} \mathbf{NSPACE}\left(n^k\right) = \mathbf{NSPACE}\left(n^{\mathcal{O}(1)}\right)
\end{aligned}$$

Nejdůležitější je třída **P**, protože algoritmy, které pracují v polynomiálním čase, jsou tradičně považovány za použitelné v praxi. Problémy v **P** se proto někdy označují jako *zvládnutelné*. Algoritmus $A \in \mathbf{TM}$ pracující v polynomiálním čase budeme jednoduše označovat jako *polynomiální algoritmus*.

Je-li $\mathcal{C}$ třída jazyků, pak $\mathbf{co}\mathcal{C}$ značíme třídu jazyků $L$, jejichž doplněk $\bar{L}$ leží v $\mathcal{C}$.

## 6. Realističtější výpočetní modely

Popsaný Turingův stroj má velmi jednoduchý popis, což je výhodné pro zkoumání jeho vlastností, ale znesnadňuje to „psaní programů“, tj. tvorbu Turingových strojů realizujících nějaký algoritmus, který máme popsán slovně, v pseudokódu nebo nějakém současném programovacím jazyce.

Předpokládejme např., že v průběhu výpočtu chceme používat počítadlo počtu dosud provedených kroků. Takové počítadlo je možné realizovat vyhrazením části pásky, což však přináší komplikace. Nevíme-li předem, jak velké číslo na počítadle budeme potřeba zapsat, budeme muset v průběhu programu prostor pro počítadlo zvětšovat. To lze posouváním symbolů realizovat, ale je to komplikace, která se skutečnou povahou algoritmu nemá nic společného.

V této kapitole proto zavedeme pohodlnější výpočetní modely. Tím ovšem vznikne otázka, vzhledem k jakému modelu uvažujeme naše složitostní třídy. Důležitým výsledkem bude, že časové i prostorové nároky jednotlivých modelů jsou svázány polynomiálně. Z toho plyne, že třída **P** je na použitém modelu nezávislá. To by neplatilo např. pro třídu algoritmů pracujících v lineárním čase. To je další důvod, proč je třída **P** hlavním předmětem teoretického zájmu.

### 6.1. Vícepáskové Turingovy stroje.

Problém s počítadlem se nejlépe vyřeší tím, že stroji přidáme jednu pásku, která bude vyhrazena jen pro počítadlo. Takto můžeme přidávat i další pásky pro jednotlivé úkoly programu. To vede k definici vícepáskového stroje. Je to stroj podobný obyčejnému Turingovu stroji opatřený $k$ páskami a $k$ hlavami. V jednom kroku se provede standardní krok Turingova stroje na všech páskách současně. V tomto případě je přirozené povolit, aby daná hlava zůstala na svém místě.

Matematicky dostáváme následující definici.

**Definice 6.1.** *$k$-páskový Turingův stroj* je čtveřice $(Q, \Sigma, q_s, P)$, kde $\Sigma$, $Q$ a $\mathcal{M}$ jsou jako v Definici 2.1 a program $P$ je nějaká množina

$$P \subseteq Q \times \Sigma^k \times Q \times \Sigma^k \times \mathcal{M}^k\,.$$

Jedna instrukce $k$-páskového stroje má tedy tvar

$$\left\langle q, \begin{pmatrix} s_1 \\ \vdots \\ s_k \end{pmatrix}, q', \begin{pmatrix} s'_1 \\ \vdots \\ s'_k \end{pmatrix}, \begin{pmatrix} m_1 \\ \vdots \\ m_k \end{pmatrix} \right\rangle.$$

Uvědomte si, proč v definici není $Q^k$, všechny hlavy mají jeden společný stav. Jeden stav stroje může kontrolovat všechny hlavy současně. Množinu $Q$ je libovolná konečná množina. Nic nebrání tomu, aby byla např. rovna $Q_1 \times Q_2 \times \cdots \times Q_l$.

Mluvíme-li o *$k$-páskovém Turingově stroji se vstupem a výstupem*, rozumíme tím $k$-páskový Turingův stroj, jehož první páska je pouze vstupní, stroj na ní nemůže přepisovat symboly (ale může se pohybovat v obou směrech), a poslední páska je výstupní, stroj může psát na prázdná políčka, ale nemůže přepisovat ani se vracet. Na zbylých $k-2$ pásek neklademe žádná omezení a nazýváme je *pracovní*.

Vícepáskový stroj budeme chápat vždy jako stroje se vstupem a výstupem. Na začátku výpočtu je vstup napsán na vstupní pásce, hlava vstupní pásky je na počátku vstupu a ostatní pásky jsou prázdné.

Časová náročnost vícepáskového stroje je definována obdobně jako u jednopáskového stroje.

Pro prostorovou složitost upřesníme Definice 5.2 a 5.3.

**Definice 6.2.** *Prostorovou složitost* problému měříme vzhledem k vícepáskovému stroji, přičemž prostorovou náročností výpočtu rozumíme počet navštívených políček na pracovních páskách.

Vstupní a výstupní páska se tedy při měření prostorové náročnosti nebere v úvahu. To umožňuje uvažovat stroje, které mají menší prostorovou náročnost, než je délka vstupu (například nulovou - to jsou tzv. konečné automaty). Bez takového přístupu by například třída **L** nedávala smysl. Všimněte si, že v definici nebylo nutné určovat počet pásek, protože více pásek lze na jedné pásce simulovat se stejnou prostorovou náročností (navíc jsou pouze oddělovače pásek, což na složitostní třídu nemá vliv díky Tvrzení 7.1 o lineárním zrychlení).

**Věta 6.3.** *Je-li nějaký problém $P$ v* $\mathbf{TIME}\left(f(n)\right)$ *pro $k$-páskový Turingův stroj, pak je v* $\mathbf{TIME}\left(\mathcal{O}(f(n)^2)\right)$ *pro jednopáskový Turingův stroj.*

*Důkaz.* Ukážeme, jak na jednopáskovém stroji simulovat běh $k$-páskového stroje. Označme jednopáskový stroj $J$ a $k$-páskový $K$.

Páska stroje $J$ bude po řadě obsahovat navštívené úseky jednotlivých pásek oddělené novým symbolem \$, včetně informace o poloze hlavy.

Má tedy podobu

$$\$ \text{ obsah 1. pásky } \$ \text{ obsah 2. pásky } \$ \ldots \$ \text{ obsah } k\text{-té pásky } \$.$$

Popíšeme simulaci jednoho kroku stroje $K$. Hlava projede celou pásku zleva doprava a zapamatuje si čtené symboly na jednotlivých páskách $(s_1, \ldots, s_k)$. Současně zkontroluje, zda se hlava na nějaké pásce nenachází na krajní pozici vlevo nebo vpravo (tedy v těsné blízkosti symbolu \$). Pokud taková situace nastane, posune celý obsah své pásky tak, aby vytvořila místo pro případný pohyb příslušné hlavy na dosud nenavštívené políčko. To je možné zvládnout jedním projetím.

Pak se hlava vrátí a cestou zpět provede příslušné instrukce, tedy změní symboly $s_1, \ldots, s_k$ na symboly $s'_1, \ldots, s'_k$ a provede pohyby $m_1, \ldots, m_k$. Celkem tedy pro realizaci jednoho kroku projede dvakrát svou pásku.

Každá z dílčích pásek má délku maximálně $f(n)$, neboť výpočet stroje $K$ skončí po $f(n)$ krocích a v každém kroku se může zaplnit maximálně jedno nové políčko. Délka použité pásky $J$ je tedy celkově maximálně $k\,f(n) + (k+1)$. K simulaci

jednoho kroku $k$-páskového stroje musí tedy stroj jednopáskový vykonat maximálně $2k(f(n)+c)$ kroků, kde $c$ je nějaká konstanta vyjadřující kroky nutné k simulaci změn na jednotlivých páskách (hlava se občas musí vrátit o jedno políčko).

Jestliže tedy stroj $K$ provede během výpočtu $f(n)$ kroků, bude počet kroků stroje $J$ ležet v $\mathcal{O}(f(n)^2)$. □

6.2. **Stroje RAM.** Největší omezení Turingových strojů v porovnání s naší představou o práci skutečných počítačů je skutečnost, že chceme-li zjistit obsah nějakého vzdáleného políčka, musí hlava projet celou vzdálenost. Skutečný počítač má přímý přístup k buňce paměti, jejíž adresu zná. Výpočetní model, který se v tomto smyslu podobá skutečnému počítači, se nazývá **RAM** (Random Access Machine).

Políčka pásky jsou v tomto modelu nahrazena *registry*, které mohou obsahovat libovolné celé číslo (**RAM** je tedy druh tzv. *registrového stroje*). Registrů je nekonečně mnoho a jsou očíslovány nezápornými celými čísly, která budeme nazývat jejich *adresy*. Nultý registr funguje jako *pracovní*, nazývá se také někdy *akumulátor*. Obsah $i$-tého registru budeme značit $[i]$.

Důležitou vlastností **RAM** je schopnost *nepřímého adresování*, tedy možnost pracovat s registrem, jehož adresa je obsahem jiného registru. To lze snadno napsat jako $[[i]]$: obsah registru, jehož adresa je obsažena v $i$-tém registru.

**RAM** má dále *vstupní a výstupní kanál*, které můžeme chápat jako zvláštní registry schopné obsahovat konečné posloupnosti celých čísel, přičemž ze vstupního kanálu je možné pouze číst (každý prvek posloupnosti nejvýše jednou a jeden po druhém), do výstupního kanálu lze pouze psát.

Zvláštním registrem je dále *počítadlo*, které řídí výpočet, hodnotu počítadla budeme označovat $\kappa$. *Výpočet* **RAM** je, podobně jako u **TM**, definován programem. *Program* **RAM** je očíslovaná posloupnost instrukcí, které můžeme rozdělit do čtyř skupin:

- vstup a výstup:

| | | |
|---|---|---|
| READ $j$ | INPUT $\to [j]$ | zapiš další prvek vstupu do registru $j$ |
| PRINT $j$ | $[j] \to$ OUTPUT | vytiskni obsah $j$-tého registru |

- komunikace pracovního registru s pamětí:

| | | |
|---|---|---|
| STORE $j$ | $[0] \to [j]$ | ulož obsah pracovního registru do $i$-tého registru |
| STORE $\uparrow i$ | $[0] \to [[i]]$ | ulož obsah pracovního registru do registru, jehož adresa je uložena v registru $i$ |
| LOAD $j$ | $[j] \to [0]$ | nahraj obsah $i$-tého registru do pracovního registru |
| LOAD $\uparrow i$ | $[[i]] \to [0]$ | nahraj do pracovního registru obsah registru, jehož adresa je uložena v registru $i$ |
| LOAD $= i$ | $i \to [0]$ | změň hodnotu pracovního registru na $i$ |

- aritmetické operace:

| | | |
|---|---|---|
| ADD $j$ | $[0] + [j] \to [0]$ | přičti obsah $i$-tého registru do pracovního registru |
| ADD $\uparrow$ $j$ | $[0] + [[j]] \to [0]$ | přičti do pracovního registru obsah registru, jehož adresa je obsažena v registru $j$ |
| SUB $j$ | $[0] - [j] \to [0]$ | odečti obsah $i$-tého registru od pracovního registru |
| SUB $\uparrow$ $j$ | $[0] - [[j]] \to [0]$ | odečti od pracovního registru obsah registru, jehož adresa je obsažena v registru $j$ |
| HALF | $\lfloor [0]/2 \rfloor \to [0]$ | odstraň nejméně významný bit pracovního registru |

- řízení toku instrukcí

| | | |
|---|---|---|
| JUMP $j$ | $\kappa := j$ | jdi na instrukci $j$ |
| HALT | $\kappa := 0$ | zastav se (lze nahradit skokem na libovolnou neexistující instrukci) |
| POS $j$ | if $[0] > 0$ then $\kappa := j$ | je-li obsah pracovního registru kladný, jdi na instrukci $j$ |
| NEG $j$ | if $[0] < 0$ then $\kappa := j$ | je-li obsah pracovního registru záporný, jdi na instrukci $j$ |
| ZERO $j$ | if $[0] = 0$ then $\kappa := j$ | je-li obsah pracovního registru nula, jdi na instrukci $j$ |

Výpočet je zahájen instrukcí jedna a vykonává instrukce programu popořadě, pokud řídící instrukce neurčí jinak. Provádí se tedy vždy instrukce $\kappa$ a každá neřídící instrukce zvětší $\kappa$ o jedna.

Také v případě **RAM** existuje celá řada různých konvencí týkajících se zejména způsobu manipulace s pamětí, které nemají na vlastnosti zásadní vliv. Důležitá je ovšem definice aritmetických operací, které stroj může provést v jednom kroku. Pokud např. dovolíme v jednom kroku násobit obsah registrů, výkon stroje se zvýší více než konstantně (pak se někdy mluví o **MRAM**). Podobně můžeme uvažovat o dalších operacích.

Délku zápisu čísla $a$ označme $d(a)$. Délka vstupu je součet délek čísel ve všech registrech, přičemž nula má nulovou délku. (Uvažujme například dyadický zápis čísel s jedním bitem rezervovaným pro znaménko.)

Operace sčítání, odčítání a dělení dvěma, které jsme dovolili v naší definici, mají tu vlastnost, že jejich „skutečná“ náročnost je lineární v délce zápisu, tedy leží v $\log k$, kde $k$ je větší z operandů.

Existují dva základní přístupy k měření složitosti výpočtu **RAM**. První z nich, nazývaná *jednotková cena* (*unit cost*), počítá pouze počet provedených instrukcí, druhá z nich, *logaritmická cena* (*logarithmic cost*), bere v úvahu i délku zpracovávaných čísel a provedení aritmetických instrukcí má cenu přímo úměrnou délce většího z operandů. Faktory logaritmické velikosti jsou tedy obzvlášť závislé na přijatém modelu, což může být zdrojem různých nedorozumění, pokud se zabýváme složitostními třídami, které nejsou dostatečně robustní. Podobně je to s otázkou začlenění násobení mezi elementární instrukce. Třída **P** je však vůči všem uvedeným možnostem invariantní.

Pro účely simulace **RAM** Turingovým strojem je nutné uvažovat logaritmickou cenu, jednotková cena není realistická. Následující tvrzení omezuje délku čísel, která se v průběhu výpočtu můžou v registrech objevit.

**Lemma 6.4.** *Po $t$ krocích stroje* **RAM** *je délka libovolného registru $d(r_i)$ nejvýše $t + d(I) + d(k)$, kde $d(I)$ je délka vstupu a $d(k)$ je délka nejdelší konstanty obsažené v programu.*

*Důkaz.* Tvrzení platí pro $t = 0$. Předpokládejme, že tvrzení platí pro $t - 1$.

- Příkazy JUMP, POS, NEG, ZERO a HALT obsahy registrů nemění.
- Příkazy LOAD a STORE přesouvají obsah z registru do registru a tvrzení pro ně tedy platí.
- Příkaz READ načítá obsah vstupu, jehož délka je v odhadu zastoupena sčítancem $d(I)$.
- Příkaz HALF obsah registru zkracuje.
- Příkazy ADD a SUB sčítají dvě celá čísla $a$ a $b$, pro něž podle indukčního předpokladu platí

  $$d(a),\, d(b) \leq t - 1 + d(I) + d(k).$$

  Stačí uvážit, že $d(a + b)$ je nejvýše $\max\{d(a), d(b)\} + 1$.

□

**Věta 6.5.** *Jestliže je nějaký problém v* $\mathbf{TIME}\,(f(n))$ *pro* **RAM**, *pak je v* $\mathbf{TIME}\left(f(n)^3\right)$ *pro 6-páskový Turingův stroj.*

*Důkaz.* Nechť $M$ je 6-páskový Turingův stroj, kterým simulujeme výpočet **RAM**. Instrukce **RAM** jsou začleněny do stavů $M$ včetně stavu počítadla.

- První páska stroje $M$ je vstupní.
- Druhá slouží k reprezentaci registrů, kromě nultého. Sestává z posloupnosti řetězců ve tvaru $b(i) : [b(i)]$, kde $b$ je nějaká permutace na množině indexů registrů, se kterými se již pracovalo. Jednotlivé registry jsou odděleny speciálním znakem.
- Na třetí pásce je zaznamenána adresa právě hledaného registru.
- Čtvrtá páska obsahuje pracovní registr a jsou na ní prováděny aritmetické operace.
- Pátá páska je výstupní.

Při hledání registru $j$ se z druhé pásky postupně načítají dvojice $b(i) : [b(i)]$. Pokud $b(i) = j$, je $[b(i)]$ zpracováno podle typu prováděné instrukce.

Při jakékoli změně registru se jeho zápis v řadě registrů smaže a zapíše znovu na konec. (Tím se vyhneme úvahám o posouvání obsahu a vytváření místa pro dlouhé registry.)

Nyní ukážeme, že časová náročnost simulace jednoho kroku **RAM** naším Turingovým strojem je $\mathcal{O}(f(n)^2)$, kde $n$ je délka vstupu.

Časově nejnáročnější je aritmetická instrukce s nepřímým adresováním, kdy je potřeba dvakrát prohledat druhou pásku (poprvé pro nalezení registru $j$ a poté registru $[j]$).

Druhá páska obsahuje $\mathcal{O}(f(n))$ dvojic (včetně smazaných úseků). Délka každé dvojice je podle předchozího lemmatu také $\mathcal{O}(f(n))$. Všimněme si zejména, že adresy použitých registrů také podléhají omezení z lemmatu. Použitý prostor pásky s registry je tedy v průběhu celého výpočtu omezen $\mathcal{O}(f(n)^2)$ a jeho prohledání vyžaduje $\mathcal{O}(f(n)^2)$ kroků stroje $M$.

Aritmetické operace (ADD, SUB, HALF) pracují s čísly délky $\mathcal{O}(f(n))$, a jejich výpočet lze tedy provést v čase $\mathcal{O}(f(n))$. Celkem tedy stroj $M$ při simulaci vykoná $\mathcal{O}(f(n)^3)$ kroků. □

## 7. Vztahy mezi třídami

**Tvrzení 7.1** (Lineární zrychlení)**.** *Nechť* $L \in \mathbf{TIME}(f(n))$ *a* $L \in \mathbf{SPACE}(g(n))$. *Pak také* $L \in \mathbf{TIME}(\varepsilon f(n))$ *a* $L \in \mathbf{SPACE}(\varepsilon g(n))$ *pro každé* $\varepsilon > 0$.

*Důkaz.* Mějme $M \in \mathbf{TM}$ odpovídající třídám $\mathbf{TIME}(f(n))$ a $\mathbf{SPACE}(g(n))$. Zkonstruujeme stroj $M'$ odpovídající třídám $\mathbf{TIME}(\varepsilon f(n))$ a $\mathbf{TIME}(\varepsilon g(n))$.

Buď $k$ přirozené číslo, které určíme později. Rychlost stroje $M'$ bude spočívat v tom, že v jednom cyklu sestávajícím z několika kroků nasimuluje $k$ kroků stroje $M$. Množina symbolů $\Sigma'$ stroje $M'$ obsahuje všechny $k$-tice symbolů $\Sigma$ stroje $M$. Tedy

$$\Sigma' = \Sigma^k.$$

Obsah pásky stroje $M'$ obsahuje v každé fázi výpočtu obsah pásky stroje $M$ seskupené do $k$-tic (předpokládáme, že i vstup je takto zadán). Prostorová náročnost stroje $M'$ je tedy $\left\lceil \frac{g(n)}{k} \right\rceil$.

Hlava stroje $M'$ čte $k$-tici, která obsahuje hlavu stroje $M$ (a pamatuje si její polohu). Po $k$ krocích stroje $M$ se tedy může změnit pouze $k$-tice čtená a její pravý nebo levý soused a změna je těmito $k$-ticemi jednoznačně zadána. Instrukce stroje $M'$ tedy vytvoříme tak, aby stroj v jednom cyklu

- přečetl levého nebo pravého souseda čteného políčka, pokud se stane, že hlava simulovaného stroje tohoto souseda v rámci uvažovaných $k$-kroků navštíví;
- zasažená políčka (odpovídající nejvýše $2k$ políčkům stroje $M$) změnil tak, aby odpovídala stavu pásky stroje $M$ po $k$ krocích;
- zastavil se v $k$-tici obsahující po $k$ krocích hlavu stroje $M$.

Snadno si rozmyslíme, že k tomu stačí tři kroky. Časová náročnost stroje $M'$ je tedy nejvýše $\left\lceil \frac{3\,f(n)}{k} \right\rceil$. Nyní stačí např. zvolit $k = \left\lceil \frac{6}{\varepsilon} \right\rceil$. □

Předchozí věta nás opravňuje k tomu, abychom ve složitostních funkcích nebrali ohled na konstanty. Můžeme tedy např. místo $\mathbf{TIME}(\mathcal{O}(f))$ psát pouze $\mathbf{TIME}(f)$

Ukazuje se, že ne všechny funkce jsou vhodné pro určování složitosti. Jak uvidíme, problém může být s funkcemi, které je obtížné spočítat.

**Definice 7.2.** Funkce $f(n)$ je *časově (prostorově) konstruovatelná*, pokud existuje Turingův stroj, který ji počítá a pro výpočet se vstupem $n$ potřebuje nejvýše $f(n)$ kroků (prostor nejvýše $f(n)$).

Všechny „přirozené“ funkce jsou konstruovatelné, definice slouží pouze k vyloučení jistých patologických případů, viz Tvrzení 7.10. Všimněme si také poněkud opatrné formulace. Pokud bychom řekli, že konstruovatelná funkce $f$ je spočitatelná s výpočetní náročností $f$, mohlo by dojít k nedorozumění. Výpočetní náročnost se totiž měří délkou vstupu, který je typicky logaritmický vzhledem k velikosti čísla. Požadavek na konstruovatelnou funkci je tedy velmi volný, při binárním zápisu vstupu má algoritmus exponenciální čas (prostor) vzhledem k velikosti vstupu. V případech, kdy chceme náročnost měřit velikostí čísla, nikoli velikostí jeho zápisu, se někdy používá unární zápis: číslo $n$ je reprezentováno vstupem $1^n$. Mohli bychom tedy říci, že $f$ je konstruovatelná funkce, pokud je spočitatelná s náročností $f$ při unárním vstupu.

Nejprve si uvědomme triviální vztahy mezi třídami.

**Tvrzení 7.3.** *Pro jednotlivé třídy složitosti platí:*

(1) $\mathbf{TIME}(f(n)) \subseteq \mathbf{SPACE}(f(n))$
(2) $\mathbf{TIME}(f(n)) \subseteq \mathbf{NTIME}(f(n))$
(3) $\mathbf{SPACE}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$

*Důkaz.* (1) Stroj $M$ nemůže za čas $f(n)$ navštívit více než $f(n)$ políček na pásce.
(2) a (3) Deterministický stroj je také nedeterministický stroj.

□

Důležitou otázkou je možnost deterministického výpočtu deterministickým strojem. Nejprve ukážeme, že nedeterministický čas lze simulovat stejně velkým deterministickým prostorem. Základní myšlenkou takové simulace bude *počítadlo* zachycující *vektor voleb*.

**Tvrzení 7.4.**

$$\mathbf{NTIME}\left(f(n)\right) \subseteq \mathbf{SPACE}\left(f(n)\right).$$

*Důkaz.* Chceme simulovat běh nedeterministického stroje $M$ nějakým deterministickým strojem $M'$. Budeme bez újmy na obecnosti předpokládat, že simulující stroj má dvě pracovní pásky. Na jedné z nich budeme simulovat výpočet stroje $M$, na druhé si budeme zaznamenávat, jaké volby mezi různými pokračováními jsme provedli. Nechť $c$ je maximální počet instrukcí aplikovatelných na jeden okamžitý popis. Můžeme pro jednoduchost předpokládat, že voleb je přesně $c$ v každém kroku (přidáme neexistující volby vedoucí k zastavení a zamítnutí vstupu).

Každý výpočet nedeterministického stroje je dán tzv. *vektorem voleb*, což je posloupnost

$$d_1, \ldots, d_t,$$

kde $t \leq f(n)$ je délka výpočtu a $d_i$ určuje, kterou instrukci z $c$ možných si stroj zvolil v $i$-tém kroku.

Druhá páska tedy bude obsahovat vektor voleb provedených při simulaci. Je-li vektor příliš krátký, znamená prázdné (nebo smazané) políčko volbu první instrukce a vektor je prodloužen o jedničku. Kdykoli simulace jedné z větví výpočtu stroje $M$ skončí, stroj $M'$ zvýší vektor voleb o jedna. Přesněji řečeno, provede následující operaci: najde největší $t'$, pro které je $d_{t'} < c$, zvýší ho o jedna a všechna $d_i$, $i > t'$ smaže. Poté simuluje výpočet nové větve $M$ dané aktuálním vektorem voleb. (Výpočet končí po simulaci, ve které jsou všechna $d_i = c$.)

Počítadlo celkem spotřebuje prostor $f(n) \log c$ a celková prostorová náročnost je tedy $\mathcal{O}(f(n)$. Nyní stačí použít větu o lineárním zrychlení. □

Pro určení vztahu nedeterministické prostorové náročnosti k deterministické je zásadní problém Dostupnost. Jeho vstupem je orientovaný graf $G(V, E)$ a dva jeho vrcholy $s$ a $t$. Máme rozhodnout, zda z $s$ do $t$ vede cesta. (Problém bývá anglicky nazýván „s-t connectivity“ nebo Stcon.)

**Lemma 7.5.** Dostupnost $\in \mathbf{SPACE}\left(\log^2(n)\right)$, *kde $n$ je počet vrcholů grafu.*

*Důkaz.* Budeme psát $x \xrightarrow{i} y$, pokud z $x$ do $y$ vede cesta délky nejvýše $2^i$. Zřejmě z $x$ vede cesta do $y$ právě tehdy, když $x \xrightarrow{\lceil \log n \rceil} y$, neboť délka nejkratší cesty mezi libovolnými dvěma vrcholy je nejvýše $n$.

Každou cestu lze rozdělit na dvě části, které se od sebe liší nejvýše o jednu. Pro $i > 0$ tedy platí tedy, že $x \xrightarrow{i} y$, právě když existuje nějaký vrchol $z$ (střed cesty), pro nějž

$$x \xrightarrow{i-1} z \ \wedge \ z \xrightarrow{i-1} y.$$

Na základě tohoto poznatku popíšeme algoritmus $A(x, y, i)$, který zjistí, zda $x \xrightarrow{i} y$. Pokud $i > 1$, algoritmus projde všechny vrcholy $z$ a zjistí, zda

$$A(x, z, i-1) \ \wedge \ A(z, y, i-1).$$

Pokud $i = 0$, zjistí, zda

$$(x, y) \in E \text{ nebo } \ x = y.$$

Hloubka rekurze algoritmu bude $\log(n)$ a prostor k uložení jedné úrovně je $k \log(n)$, kde $k$ je nějaká konstanta (délka zápisu jednoho vrcholu je nejvýše $\lceil \log(n) \rceil$). Tedy celková prostorová náročnost je v $\mathcal{O}(\log^2(n))$. □

**Tvrzení 7.6** (Savitchova věta)**.** *Buď $f$ prostorově konstruovatelná funkce, $f(n) \geq \log n$. Pak*

$$\mathbf{NSPACE}\left(f(n)\right) \subseteq \mathbf{SPACE}\left(f(n)^2\right).$$

*Důkaz.* Nechť $L \in \mathbf{NSPACE}\left(f(n)\right)$ a nechť $M \in \mathbf{TM}$ je (nedeterministický) Turingův stroj, který $L$ rozhoduje. Chceme ukázat, že existuje deterministický stroj rozhodující $L$ v prostoru $f(n)^2$.

Klíčem k důkazu je fakt, že pro dané $x$ platí $x \in L$, právě když existuje posloupnost snímků stroje $M$, která definuje přijímající výpočet pro vstup $x$. Abychom toto využili, uvažujme orientovaný graf $G_S$, jehož vrcholy jsou **všechny možné** snímky stroje $M$ a z vrcholu $\sigma_1$ vede hrana do vrcholu $\sigma_2$ právě tehdy, když

$$\sigma_1 \ \xrightarrow{\delta_M} \ \sigma_2,$$

neboli pokud $\sigma_2$ je snímek, který vznikne ze snímku $\sigma_1$ po jednom kroku stroje $M$. Můžeme předpokládat, že existuje právě jeden akceptující snímek, označme ho „ANO“. Snímek se vstupem $x$ před začátkem výpočtu označme „vstup $x$“.

Nyní tedy platí $x \in L$ právě tehdy, pokud v grafu $G_S$ existuje cesta z vrcholu „vstup $x$“ do vrcholu „ANO“. Problém dostupnosti je přitom řešitelný v prostoru $\log^2 n$, kde $n$ je počet vrcholů grafu.

Počet vrcholů grafu je dán

- velikostí množiny stavů $Q_M$ stroje $M$,
- počtem možných slov zapsaných na pracovní pásce délky nejvýše $f(n)$: $|\Sigma|^{f(n)}$,
- počtem poloh hlavy na pracovní pásce: $f(n)$,
- počtem poloh hlavy na vstupní pásce: $n$.

Celkem je tedy počet vrcholů $|V| = nf(n)\,|Q_M|\,|\Sigma|^{f(n)}$ a

$$\log|V| = \log n + \log(f(n)) + \log|Q_M| + f(n)\log|\Sigma|,$$

což je pro $f(n) \geq \log n$ v $\mathcal{O}(f(n))$.

Existenci cesty je proto možné zjistit v prostoru $\mathcal{O}(f(n)^2)$.

Zbývá vysvětlit, jakým způsobem bude deterministický stroj s grafem $G_S$ pracovat. Je zřejmé, že ho nemůže celý vypsat, protože tím by překročil svůj pracovní prostor. Procházení všech vrcholů (snímků) se bude dít v maximo-lexikografickém pořadí. Můžeme bez újmy na obecnosti uvažovat všechny posloupnosti patřičné délky, které ani nemusí být zápisem nějakého snímku. (Zde používáme předpoklad prostorové konstruovatelnosti funkce $f$. Je-li $f$ prostorově konstruovatelná, stroj $M'$ umí spočítat délku snímku, aniž by překročil svou vlastní prostorovou náročnost.)

Ověřování existence hrany mezi vrcholy $\sigma_1$ a $\sigma_2$ se provede simulací jednoho kroku příslušného nedeterministického stroje. □

Simulace prostorové složitosti je časově náročná:

**Tvrzení 7.7.** *Nechť $f(n) \geq \log n$. Pak*

$$\mathbf{SPACE}\left(f(n)\right) \subseteq \bigcup_{k>1} \mathbf{TIME}\left(k^{f(n)}\right).$$

*Důkaz.* Nechť $M \in \mathbf{DTM}$ je stroj rozhodující jazyk $L$ s prostorovou náročností $f(n)$. Počet snímků je (podle diskuse v důkazu Savitchovy věty) nejvýše $k^{f(n)}$ pro nějaké $k$. Pokud stroj provede více než $k^{f(n)}$ kroků, znamená to, že nějaký snímek se objevil dvakrát, a stroj se tedy zacyklil, protože je deterministický. To je spor s předpokladem, že rozhoduje $L$. □

Kombinací Tvrzení 7.4 a Tvrzení 7.7 dostáváme

$$\mathbf{NTIME}\left(f(n)\right) \subseteq \mathbf{SPACE}\left(f(n)\right) \subseteq \bigcup_{k>1} \mathbf{TIME}\left(k^{f(n)}\right).$$

Žádná podstatně lepší než exponenciální simulace nedeterministického času deterministickým není známa.

Následující věty ukazují, že hierarchie tříd složitosti je dosti hustá.

**Tvrzení 7.8** (O prostorové hierarchii)**.** *Nechť $f(n)$ a $g(n)$ jsou funkce takové, že $\log n \leq f(n)$, $f(n) \in o(g(n))$ a $g(n)$ je prostorově konstruovatelná. Pak* $\mathbf{SPACE}\left(f(n)\right)$ *je vlastní podmnožina* $\mathbf{SPACE}\left(g(n)\right)$.

*Důkaz.* Zkonstruujeme jazyk, který leží ve třídě $\mathbf{SPACE}\left(g(n)\right)$ a neleží ve třídě $\mathbf{SPACE}\left(f(n)\right)$. Konstrukce má podobu diagonálního argumentu, tj. využívá myšlenky běhu stroje, který má na vstupu svůj vlastní kód. Uvažujme nějaké kódování Turingových strojů a označme $[M]$ kód stroje $M$. Abychom mohli manipulovat s délkou $[M]$, budeme uvažovat slova $1^m0[M]$, tzv. *prefixové kódy* stroje $M$. Je zřejmé, že z každého prefixového kódu lze snadno zrekonstruovat kód původní.

Slíbený jazyk $L$ definujeme pomocí stroje $U$, který ho rozhoduje a pracuje v prostoru $g(n)$. Stroj $U$ je jistou formou univerzálního Turingova stroje. Základní myšlenka důkazu je následující: Stroj $U$ je schopen v prostoru $g(n)$ rozpoznat, jaký bude výstup stroje $M$ pracujícího v prostoru $f(n)$ včetně informace, zda výpočet diverguje. Do rozporu s nerozhodnutelností problému HaltingProblem se nedostáváme díky tomu, že $U$ má garantováno prostorové omezení simulovaného stroje. Pokud simulace probíhá příliš dlouho, pozná stroj $U$, že se simulovaný stroj zacyklil, díky svému dodatečnému prostoru, který využije pro počítadlo.

Stroj $U$ tedy postupuje takto. Na vstupu $x$ rozhodne (v konstantním prostoru), zda je $x$ prefixový kód nějakého $M$, a pokud ano, vymezí si na pásce prostor $\frac{1}{2}g(n)$, pro simulaci pracovního prostoru stroje $M$ a počítadlo kroků délky $\frac{1}{2}g(n)$ (tedy s rozsahem $2^{\frac{1}{2}g(n)}$ kroků). (Zde používáme předpoklad prostorové konstruovatelnosti!)

Stroj $U$ dále simuluje výpočet stroje $M$ na vstupu $x$, tedy $M(1^m0[M])$, a počítá jeho kroky. Výstup stroje $U$ je dán následujícími pravidly:

(a) pokud simulace přesáhla vymezený prostor, přijmi;
(b) pokud simulace provedla počet kroků stroje $M$ přesahující rozsah počítadla, přijmi;
(c) pokud simulace skončí bez komplikací, tedy aniž by nastalo (a) nebo (b), odpověz opačně než $M$.

Předpokládejme nyní, že nějaký stroj $M$, který pracuje v prostoru $f(n)$, rozhoduje $L = L(U)$. Chceme dojít ke sporu, a proto hledáme vstup, na kterém se odpověď $U$ a $M$ bude lišit. Takovým vstupem bude dostatečně dlouhý prefixový kód stroje $M$. Uvažujme tedy vstup $x = 1^k0[M]$, kde $k$ zvolíme tak, aby

$$g(|x|) > 8 \cdot f(|x|) \log |[M]|,$$

což je možné díky $f(n) \in o(g(n))$.

Všimněme si, že prostor vymezený strojem $U$ pro simulaci je dostatečný na simulaci celého výpočtu stroje $M$ včetně binární reprezentace symbolů $\Sigma_M$ a stavů $Q_M$.

Protože $M$ konverguje, platí také, že počet kroků jeho výpočtu je menší než počet všech jeho možných snímků, tedy (viz diskusi v důkazu Savitchovy věty) méně než

$$r = |x|\, f(|x|)\, |Q_M|\, |\Sigma_M|^{f(|x|)}\,.$$

Z $f(n) \geq \log n$ dostáváme

$$\log(r) < 4f(|x|) \log |[M]| < \frac{1}{2} g(|x|).$$

Z toho plyne, že výstup stroje $U$ se bude řídit pravidlem (c), což ukazuje, že $M$ nepřijímá $L$, spor. □

Podobný důkaz následující věty vynecháváme.

**Tvrzení 7.9** (O časové hierarchii)**.** *Nechť $f(n)$ a $g(n)$ jsou funkce takové, že $n \leq f(n)$, $f(n)\log(f(n)) \in o(g(n))$ a $g(n)$ je časově konstruovatelná. Pak* $\mathbf{TIME}\,(f(n))$ *je vlastní podmnožina* $\mathbf{TIME}\,(g(n))$.

Následující věta, která platí i pro časovou složitost, se zdá být v rozporu s Větou 7.8. To ukazuje na důležitost chybějícího předpokladu o prostorové konstruovatelnosti.

**Tvrzení 7.10** (O mezeře)**.** *Pro libovolnou rekursivní funkci $r > n$ existuje rostoucí rekursivní funkce $f(n)$, pro kterou platí*

$$\mathbf{SPACE}\,(f(n)) = \mathbf{SPACE}\,(r(f(n)))\,.$$

To tedy např. znamená, že existuje rekursivní funkce $f(n)$ taková, že třída $\mathbf{SPACE}\,(f(n))$ je identická s třídou $\mathbf{SPACE}\left(2^{2^{f(n)}}\right)$.

*Důkaz.* Inkluze $\mathbf{SPACE}\,(f(n)) \subseteq \mathbf{SPACE}\,(r(f(n)))$ je zřejmá. Ukážeme, že může nastat i opačná inkluze.

Chceme, aby každý stroj $M \in \mathbf{DTM}$ pracoval v prostoru menším než $f(n)$, nebo větším než $r(f(n))$. Určíme hodnotu $f(n)$.

Nechť $G$ je efektivní číslování $\mathbf{DTM}$. Označme $s(i,x) \in \mathbb{N} \cup \{\infty\}$ prostor spotřebovaný při výpočtu $G(i)(x)$. Hodnota $\infty$ se nabývá, pokud výpočet diverguje (a to i tehdy, pokud se stroj zacyklí, a celý nekonečný výpočet je omezen na konečný prostor).

Hodnota $f(n)$ bude nejmenší přirozené číslo takové, aby platilo

- $f(n) > f(n-1)$ (pro $n > 1$),
- pro každý vstup $x$ délky $n$ a každé $i \leq |x| = n$ platí, že buď $s(i,x) \leq f(n)$, nebo $s(i,x) > r(f(n))$.

Takové číslo existuje, protože množina hodnot $s(i,x)$ je konečná a existuje nekonečně vzájemně disjunktních intervalů $(i, r(i)]$. Stačí uvažovat intervaly

$$I_j = (r^j(1), r^{j+1}(1)],$$

kde $r^1(1) = r(1)$ a $r^{j+1}(1) = r(r^j(1))$.

Nechť nyní $L \in \mathbf{SPACE}\,(r(f(n)))$. Jazyk $L$ je tedy rozhodován nějakým strojem $G(k)$ tak, že pro libovolné $x$ je $s(k,x) \leq r(f(|x|))$. Je-li $k \leq |x|$, plyne z definice funkce $f$, že $s(k,x) \leq f(|x|)$. Modifikujeme-li stroj $G(k)$ tak, aby vstupy délky méně než $k$, kterých je konečně mnoho, rozhodoval tabulkou uloženou v programu, tedy s nulovou prostorovou náročností, dostáváme $L \in \mathbf{SPACE}\,(f(n))$.

Zbývá ověřit, že funkce $f(n)$ je rekursivní. Popíšeme stroj $M$, který ji počítá. Předpokládejme, že hodnotu $f(n-1)$ lze spočítat (položme $f(0) = 0$). Stroj $M$ postupně probírá všechna čísla $t$ větší než $f(n-1)$. Pro každé $t$, každé $i \leq n$ a každé $x$ délky $n$ bude simulovat výpočet $G(i)(x)$ a současně měřit prostor spotřebovaný simulovaným výpočtem stroje $G(i)$ na vstupu $x$, dokud nenastane jedna z následujících situací

- výpočet skončil,
- výpočet se zacyklil (to lze zjistit kontrolou počtu kroků),
- prostor použitý strojem $G(i)$ přesáhl $r(t)$.

Pokud výpočet skončil a použitý prostor je v intervalu $(t, r(t)]$, stroj $M$ začne postup znovu s $t+1$. □

**Důsledek 7.11.** *Funkce $f(n)$ z předchozí věty není prostorově konstruovatelná.*

## 8. Booleovské formule a obvody

**Definice 8.1** (Booleovská formule)**.** Uvažujme abecedu

$$\Sigma = X \cup \{\wedge, \vee, \neg\} \cup \{(,)\},$$

kde $X$ je nějaká množina proměnných (obvykle $X = \{x_i \mid i \in \mathbb{N}\}$). Prvek $\varphi \in \Sigma^+$ se nazývá *booleovská formule*, právě když je splněna jedna z následujících podmínek.

(1) $\varphi \in X$
(2) $\varphi = (\neg\psi)$, kde $\psi$ je booleovská formule,
(3) $\varphi = (\psi_1 \vee \psi_2)$, kde $\psi_1$ a $\psi_2$ jsou booleovské formule,
(4) $\varphi = (\psi_1 \wedge \psi_2)$, kde $\psi_1$ a $\psi_2$ jsou booleovské formule.

Nehrozí-li nedorozumění, budeme při zápisu booleovských formulí často vynechávat závorky.

Formuli $\varphi$ nazýváme *pozitivní literál* pokud $\varphi \in X$ a *negativní literál* pokud $\varphi = \neg x$, kde $x \in X$.

Symbolem $X|_\varphi$ značíme množinu promněnných, které se vyskytují ve $\varphi$.

Libovolné zobrazení $h : X|_\varphi \longrightarrow \{0, 1\}$ nazveme *ohodnocením formule $\varphi$*.

**Definice 8.2.** Řekneme, že booleovská formule $\varphi$ je ohodnocením $h$ *splněna*, píšeme $h(\varphi) = 1$, právě když platí jedna z následujících podmínek.

(1) $\varphi = x$, $x \in X$ a $h(x) = 1$,
(2) $\varphi = \neg\psi$ a $\psi$ ohodnocením $h$ splněna není,
(3) $\varphi = (\psi_1 \vee \psi_2)$ a alespoň jedna z formulí $\psi_1$, $\psi_2$ je ohodnocením $h$ splněna,
(4) $\varphi = (\psi_1 \wedge \psi_2)$ a obě formule $\psi_1$ a $\psi_2$ jsou ohodnocením $h$ splněny.

**Definice 8.3.** Řekneme, že formule je *splnitelná*, pokud existuje ohodnocení, které ji splňuje.

Nesplnitelná formule se nazývá *spor*. Formule, která je splněna každým ohodnocením se nazývá *tautologie*. Formule $\psi_1$ a $\psi_2$ jsou *ekvivalentní* (značíme $\equiv$), pokud pro každé ohodnocení $h$ platí, že $h$ splňuje $\psi_1$, právě když splňuje $\psi_2$.

Je známo, že booleovské formule mají následující vlastnosti: *Komutativita*:

$$\psi_1 \vee \psi_2 \equiv \psi_2 \vee \psi_1,$$
$$\psi_1 \wedge \psi_2 \equiv \psi_2 \wedge \psi_1,$$

*asociativita*:

$$((\psi_1 \vee \psi_2) \vee \psi_3) \equiv (\psi_1 \vee (\psi_2 \vee \psi_3)),$$
$$((\psi_1 \wedge \psi_2) \wedge \psi_3) \equiv (\psi_1 \wedge (\psi_2 \wedge \psi_3)),$$

zákon *vyloučení třetího* (neboli zákon *negace negace*):

$$\neg\neg\psi \equiv \psi,$$

*idempotence*:

$$\varphi \vee \varphi \equiv \varphi,$$
$$\varphi \wedge \varphi \equiv \varphi,$$

distributivita:

$$((\psi_1 \vee \psi_2) \wedge \psi_3) \equiv ((\psi_1 \wedge \psi_3) \vee (\psi_2 \wedge \psi_3)),$$

$$((\psi_1 \wedge \psi_2) \vee \psi_3) \equiv ((\psi_1 \vee \psi_3) \wedge (\psi_2 \vee \psi_3))$$

a *De Morganova pravidla*:

$$\neg\psi_1 \vee \neg\psi_2 \equiv \neg(\psi_1 \wedge \psi_2),$$

$$\neg\psi_1 \wedge \neg\psi_2 \equiv \neg(\psi_1 \vee \psi_2).$$

Libovolnou disjunkci literálů nazýváme *klausule*. Řekneme, že $\varphi$ je v *konjuktivní normální formě*, pokud je konjunkcí klausulí. Řekneme, že $\varphi$ je v *disjunktivní normální formě*, pokud je disjunkcí formulí, z nichž každá je konjunkcí literálů.

**Definice 8.4.** Každé zobrazení $f : \{0,1\}^n \to \{0,1\}$ nazveme *booleovskou funkcí*.

**Definice 8.5.** Každá booleovská formule $\varphi$, definuje booleovskou funkci $f_\varphi : \{0,1\}^n \to \{0,1\}$, kde $n$ je počet proměnných obsažených ve $\varphi$ a $f_\varphi(a_1, a_2, \ldots, a_n) = 1$, právě když je $\varphi$ splněna ohodnocením $h : x_i \mapsto a_i$.

**Tvrzení 8.6.** *Ke každé booleovské funkci $f$ existuje booleovská formule $\varphi$ taková, že $f = f_\varphi$.*

*Důkaz.* Nechť $H_+$ je množina takových ohodnocení proměnných $\{x_1, \ldots, x_n\}$, že

$$f(h(x_1), \ldots, h(x_n)) = 1.$$

K danému ohodnocení $h$ definujeme formuli

$$\psi_h = \bigwedge_{i=1}^{n} l_{h,i},$$

kde

$$l_{h,i} = \begin{cases} x_i, & \text{pokud } h(x_i) = 1 \\ \neg x_i, & \text{pokud } h(x_i) = 0. \end{cases}$$

Formuli $\varphi$ nyní definujme jako disjunkci

$$\varphi = \bigvee_{h \in H_+} \psi_h.$$

Přímočarou úvahou je možno ověřit, že skutečně $f = f_\varphi$. □

**Tvrzení 8.7.** *Každá formule $\varphi$ je ekvivalentní nějaké formuli v konjuktivní normální formě a nějaké formuli v disjunktivní normální formě.*

*Důkaz.* Uvažme booleovskou funkci $f$ definovanou formulí $\varphi$. Konstrukcí popsanou v předchozí větě dostaneme formuli v disjunktivní normální formě, která je s $\varphi$ ekvivalentní.

Pro získání konjunktivní normální formy formule $\varphi$ vezměme disjunktivní normální formu formule $\neg\varphi$

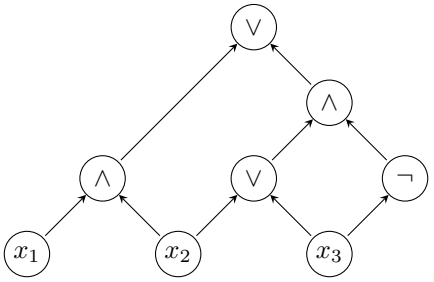
$$\neg\varphi = \bigvee_{j=1}^{m} (\bigwedge_{i=1}^{n} y_{i,j}),$$

kde $y_{i,j}$ jsou literály. Použitím De Morganových pravidel a pravidla negace negace dostaneme konjunktivní normální formu formule $\varphi$:

$$\varphi = \neg(\bigvee_{j=1}^{m} (\bigwedge_{i=1}^{n} y_{i,j})) = \bigwedge_{j=1}^{m} (\neg(\bigwedge_{i=1}^{n} y_{i,j})) = \bigwedge_{j=1}^{m} (\bigvee_{i=1}^{n} (\neg y_{i,j})).$$

□

*Cyklus* v orientovaném grafu $G(V, E)$ je posloupnost vrcholů $(v_0, \ldots, v_n)$, $n \geq 1$, takových, že $(v_i, v_{(i+1) \bmod n}) \in E$. Acyklický graf je graf, který neobsahuje cyklus.




Obrázek 2. Booleovský obvod odpovídající formuli $(x_1 \wedge x_2) \vee ((x_2 \vee x_3) \wedge \neg x_3)$.

**Definice 8.8.** Booleovský obvod je koněčný orientovaný acyklický graf $G = (V, E)$, ve kterém je každý vrchol označen jedním prvkem množiny $\{x_1, \ldots, x_n\} \cup \{\vee, \wedge, \neg\} \cup \{0, 1\}$ a platí:

(1) Vrcholy typu $x_i$ a 0, 1 jsou zdrojové
(2) Vrchol typu $\neg$ má právě jednu vstupní hranu
(3) Vrchol typu $\vee$ nebo $\wedge$ má právě dvě vstupní hrany

**Definice 8.9.** Vrchol v booleovském obvodu má při ohodnocení $h$ hodnotu, která je dána induktivně pomocí jeho vstupních vrcholů a pomocí jeho typu.

**Definice 8.10.** Vrchol $v$ počítá booleovskou funkcí $f : \{0,1\}^n \to \{0,1\}$, kde $f(a_1, \ldots, a_n)$ je dáno hodnotou vrcholů $v$ při ohodnocení $h : x_i \mapsto a_i$.

Obvykle budeme předpokládat, že uvažovaný booleovský obvod má právě jeden výstupní vrchol (vrchol z něhož nevede hrana). Tento vrchol máme na mysli, pokud mluvíme o funkci počítané příslušným obvodem.

Srovnáním Definice 8.1 a Definice 8.8 vidíme, že booleovský obvod je jiným vyjádřením booleovské formule, a naopak.

**Definice 8.11.** *Velikostí booleovské formule* rozumíme celkový počet výskytů $\vee$, $\wedge$ nebo $\neg$ ve formuli. *Velikostí booleovského obvodu* rozumíme počet jeho vrcholů typu $\vee$, $\wedge$ nebo $\neg$.

Booleovský obvod na Obrázku 2 má tedy velikost 5, stejně jako formule, které odpovídá.

**Tvrzení 8.12.** *Každou booleovskou funkci $f\{0,1\}^n \to \{0,1\}$ lze vypočítat booleovským obvodem velikosti nejvýše $2^{n-1}n + n - 1$.*

*Důkaz.* Použijeme obvod odpovídající formuli počítající funkci $f$, která je buď v disjunktivní normální formě, nebo konjunktivní normální formě (viz Tvrzení 8.7). Pokud je $|\{x \mid f(x) = 1\}| \leq |\{x \mid f(x) = 0\}|$, použijeme disjunktivní normální formu, jinak konjunktnivní.

Uvažme např. druhý případ. Konjunktivní normální forma je konjunkcí nejvýše $2^{n-1}$ klausulí, což odpovídá $2^{n-1} - 1$ symbolům $\wedge$. Každá klausule má přesně $n$ literálů, tedy celkem nejvýše $2^{n-1}(n-1)$ symbolů $\vee$. Na všechny výskyty jedné negované proměnné stačí jediná negace.

Dostáváme obvod velikosti nejvýše

$$2^{n-1} - 1 + 2^{n-1}(n-1) + n = 2^{n-1}n + n - 1.$$

□

## 9. Redukce a úplnost

*Turingův stroj s orákulem* $A$, kde $A$ je nějaký jazyk, je vícepáskový Turingový stroj, značíme ho $M^A$, s jednou zvláštní *dotazovací páskou* a třemi zvláštními stavy `dotaz`, `ano` a `ne`. Pokud se během výpočtu dostane stroj do stavu `dotaz`, spočívá další krok pouze v přechodu do stavu `ano` nebo `ne`, podle toho, zda slovo napsané na dotazovací pásce leží, nebo neleží v $A$.

Stroj $M^A$ tedy rozhoduje jazyk $A$ v jednom kroku (je-li vstup na dokazovací pásce). Můžeme to chápat jako existenci podprogramu rozhodujícího $A$, jehož výpočet se do náročnosti $M$ nezapočítává (započítává se jen jeho spuštění, jako jeden krok). To je podobné např. rozhodnutí, že násobení budeme chápat v **RAM** jako jedinou operaci. V tomto případě ovšem může být $A$ podle definice libovolné, i nerekurzivní.

Stroje s orákulem se hodí pro zkoumání relativní obtížnosti problémů, argumentům používajícím orákula se proto říká „relativizace“. Náročnost stroje $M^A$, který rozhoduje $B$, odpovídá na otázku, jak složité je rozhodnout $B$, pokud zanedbáme nároky na rozhodování $A$. Je-li pro $M^A$ rozhodování $B$ snadné, můžeme říct, že jsme obtížnost $B$ redukovali na obtížnost $A$. To vede k následující definici.

**Definice 9.1** (Turingova (Cookova) redukce)**.** Řekneme, že jazyk $B$ je *turingovsky redukovatelný* na jazyk $A$, píšeme $B \leq_T A$, pokud existuje $M^A$, který rozhoduje $B$ v polynomiálním čase.

Tato definice chápe „snadnost“ z předchozí diskuse jako rozhodnutelnost v polynomiálním čase. Bylo by samozřejmě možné uvažovat i jiné možnosti. Tvrzení $B \leq_T A$ chápeme tak, že $B$ je lehčí (přesněji, není o mnoho těžší) než $A$.

Jiná, přísnější forma redukce dovoluje stroji $M$ použít služby orákula jen jednou, a to na konci výpočtu tak, že odpověď orákula je současně odpovědí stroje $M$. V takovém případě obvykle nemluvíme o orákulu, ale říkáme, že stroj $M$ redukuje instanci problému $B$ na instanci problému $A$ (stroj se v takovém případě pro názornost nazývá obvykle $R$). Dostáváme tak následující definici.

**Definice 9.2** (Karpova (Levinova, many-one) redukce)**.** Řekneme, že jazyk $B$ je *karpovsky redukovatelný* na jazyk $A$, píšeme $B \leq_P A$, pokud existuje polynomiální stroj $R$ takový, že $R(x) \in A$, právě když $x \in B$.

Karpova redukce opět počítá s polynomiálním časem nutným pro redukci, přesněji bychom tedy měli mluvit o *polynomiální Karpově redukci*. Název „many-one redukce“ odkazuje na fakt, že narozdíl od Turingovy redukce může stroj $R$ během mnoha svých kroků použít orákulum jen jednou. Karpova redukce byla definována v souvislosti s třídou **NP** a má zjevně smysl jen pro problémy, které jsou těžší než polynomiální. Redukovat karpovsky např. nějaký problém z **NL** nebo z **P** není zajímavé, protože ho můžeme, namísto redukce, v polynomiálním čase rovnou vyřešit. Proto se často používá ještě přísnější redukce.

**Definice 9.3** (Log-space redukce)**.** Řekneme, že jazyk $B$ je *log-space redukovatelný* na jazyk $A$, píšeme $B \leq_L A$, pokud existuje $R$ s logaritmickou prostorovou náročností takový, že $R(x) \in A$, právě když $x \in B$.

Následující věta potvrzuje, že je naplněna intuice, se kterou jsme redukce definovali: je-li $B$ redukovatelné na $A$, není $B$ těžší než $A$. Háček v této úvaze by mohl spočívat v tom, že obtížnost $B$ bude skryta právě v redukci, jako např. v případě polynomiální redukce jazyka z **L**.

**Tvrzení 9.4** (**P** je uzavřena na Karpovu redukci.)**.** *Pokud $B \leq_P A$, kde $A \in \mathbf{P}$, pak $B \in \mathbf{P}$.*

*Důkaz.* Nechť $R \in \mathbf{P}$ je Karpova redukce jazyka $B$ na $A \in \mathbf{P}$, náročnosti $p(n)$. Nechť $M$ je polynomiální algoritmus složitosti $q(n)$ rozhodující $L_2$. Pak složení $T = M \circ R$ rozhoduje $B$. Složitost $T$ je nejvýše $p(n) + q(p(n))$, protože délka $R(x)$ je nejvýše $p(|x|)$. Tím je důkaz u konce, neboť $p(n) + q(p(n))$ je polynom. □

Otázka log-space uzavřenosti **L** v následující větě je komplikovanější tím, že výstup stroje se nezapočítává do prostorové náročnosti, přičemž může být podstatně větší než $\log(n)$. Prostorová náročnost prostého složení, tedy vůbec nemusí být logaritmická.

**Tvrzení 9.5** (**L** je uzavřena na log-space redukci.)**.** *Pokud $B \leq_L A$, kde $A \in \mathbf{L}$, pak $B \in \mathbf{L}$.*

*Důkaz.* Nechť $R \in \mathbf{L}$ je log-space redukce jazyka $B$ na $A \in \mathbf{L}$ a nechť $M$ rozhoduje $A$ v logaritmickém prostoru. Pak $B$ je rozhodován následujícím algoritmem $T$ s logaritmickou prostorovou náročností. Algoritmus $T$ simuluje v na jedné pásce běh algoritmu $M$. Kdykoli $M$ chce přečíst $i$-tý symbol svého vstupu simuluje $T$ na druhé pásce běh stroje $R$. Namísto psaní výstupu však pouze na třetí pásce počítá kolikátý symbol výstupu má být vypsán. Po dosažení $i$-tého symbolu ho předá stroji $M$ jako $i$-tý symbol vstupu.

Na prvních dvou páskách je prostorová náročnost logaritmická podle předpokladu. Zbývá uvážit náročnost počítadla. Počítadlo musí obsáhnout nejvýše počet kroků stroje $R$. Ten však pracuje v logaritmickém prostoru, a počet jeho kroků je tedy v $\mathcal{O}(n^k)$ (podle obvyklé diskuse o počtu snímků). To znamená, že počítadlu stačí prostor $\mathcal{O}(\log(n))$. □

Pojem turingovské redukce byl definován Cookem v roce 1971 a pojem karpovské redukce Karpem v roce 1972. V obou případech byla motivací redukce všech problémů z **NP** na jeden z nich. To vede k definici *úplných* a *těžkých* jazyků dané složitostní třídy vzhledem k dané redukci.

**Definice 9.6.** Nechť $r \in \{T, L, P\}$. Řekneme, že jazyk $A$ je $\leq_r$*-těžký pro* $C$, pokud $B \leq_r A$ pro každý jazyk $B \in C$. Je-li navíc $A \in C$, nazývá se $\leq_r$*-úplný pro* $C$.

Obvykle se charakteristika těžkých a úplných jazyků zkracuje. Není-li řečeno jinak, nazýváme **NP**-úplným (resp. **NP**-těžkým) jazyk, který je $\leq_P$-úplný (resp. $\leq_P$-těžký) pro **NP**. V případě třídy **P** nebo **NL** je uvažovanou redukcí $\leq_L$.

## 10. Alternativní charakteristika třídy NP

Řekneme, že relace $R$ je *polynomiálně omezená*, pokud existuje polynom $p$ takový, že pro všechna $(x, y) \in R$ platí $|y| \leq p(|x|)$.

**Tvrzení 10.1.** *Jazyk $L$ leží v* **NP** *právě když existuje polynomiálně omezená relace $R_L \in \mathbf{P}$ taková, že $x \in L$, právě když $(x, y) \in R_L$ pro alespoň jedno $y$.*

Relaci $R_L$ nazýváme *svědeckou relací* jazyka $L$ a je-li $(x, y) \in R_L$, říkáme, že $y$ je *svědek* pro $x$.

*Příklad* 10.2.

- $L = \{n \mid n \text{ je složené číslo}\}$
  $R_L = \{(a, b) \mid 2 \leq b < a,\ b|a\}$
- $R_{\text{SAT}} = \{(\varphi, k) \mid\ k \text{ splňuje } \varphi\}$
- $R_{\text{CIRCSAT}} = \{(C, \sigma) \mid C(\sigma) = 1\}$.

*Důkaz.* Chceme ukázat, že charakterizace **NP** pomocí svědeckých relací je ekvivalentní definici pomocí nedeterministických strojů.

Nechť je $L$ rozhodován nedeterministickým strojem $M$. Pak jako svědeckou relaci můžeme zvolit

$$R_L = \{(x, y) \mid x \in L,\ y \text{ je vektor voleb přijímajícího výpočtu } x \text{ strojem } M\}.$$

Pokud je naopak $R_L$ svědecká relace pro $L$ rozhodovaná strojem $T$, sestrojíme nedeterministický stroj rozhodující $L$ takto:

- zvol svědka $y$ (příslušné délky dané polynomiální omezeností $R_L$),
- zjisti pomocí $T$, zda $(x, y) \in R_L$.

□

Nedeterministický stroj $T$ v předchozím důkazu se vyznačuje tím, že veškerý nedeterminismus je koncentrován na začátek výpočtu, do volby svědka, a poté je výpočet deterministický. Můžeme to také chápat tak, že stroj si na počátku nageneruje vektor voleb, kterým se potom (deterministicky) řídí při rozhodování jakou použít instrukci.

## 11. NP-úplnost

Třída **NP** obsahuje stovky úplných problémů, včetně mnoha důležitých a přirozených obtížných problémů (např. problém obchodního cestujícího, barvení grafu, hamiltonovská cesta v grafu).

Všimněme si nejprve, že relace $\leq_P$ je tranzitivní. Důkaz je snadný a podobný jako důkaz uzavřenosti **P** na $\leq_P$. Z toho plyne, že pokud nějaký **NP**-úplný problém karpovsky redukujeme na problém $B$, dokážeme tím, že i $B$ je **NP**-úplný. To je obvyklý postup při dokazování **NP**-úplnosti. První a nejdůležitější problém, stojící v čele všech podobných redukcí je problém splnitelnosti booleovských formulí nebo obvodů. Ten tak hraje pro **NP**-úplnost podobnou roli jako HaltingProblem pro nerozhodnutelnost. Problém SAT požaduje rozhodnout, zda je daná booleovská formule rozhodnutelná. Tedy

$$\text{SAT} = \{\varphi \mid \varphi \text{ je splnitelná booleovská formule}\}.$$

**Věta 11.1** (Cookova-Levinova)**.** SAT *je* **NP***-úplný.*

*Důkaz.* Ukažme nejprve, že SAT $\in$ **NP**. Nedeterministický algoritmus rozhodující SAT pracuje následovně:

- Zvol ohodnocení $h(n)$
- Pokud je $\varphi$ ohodnocením $h(n)$ splněna, přijmi; jinak odmítni.

Spočítat hodnotu formule s daným ohodnocením je jistě možné v polynomiálním čase. Uvedený algoritmus tedy pracuje v polynomiálním nedeterministickém čase a přijímá právě splnitelné formule.

Nechť nyní $A \in$ **NP** a nechť $M \in$ **NTM** rozhoduje $A$. Popíšeme, jak pro vstup $x$ utvořit formuli $\varphi(x)$, která bude splnitelná, právě když bude existovat výpočet stroje $M$, přijímající $x$.

Budeme předpokládat, že $M$ je jednopáskový stroj s jednostranně nekonečnou páskou, tedy s políčky očíslovanými přirozenými čísly. Takový předpoklad jistě není na újmu obecnosti. Nechť $M$ pracuje s abecedou $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_\ell\}$, včetně $\sigma_1 = \square$, a množinou stavů $\{q_1, q_2, \ldots, q_r\}$, přičemž $q_1$ je počáteční stav a $q_r$ stav přijímající. Doba běhu stroje $M$ nechť je omezena polynomem $p$. Na vstupu $x$ tedy stroj vykoná nejvýše $\tau := p(|x|)$ kroků a navštíví nejvýše prvních $\tau$ políček. Označme $P_M$ program stroje $M$, přičemž pohyb hlavy v instrukci budeme vyjadřovat číslem $m \in \{-1, 0, 1\}$. Po zastavení stroje jsou všechny následující snímky, až do času $\tau$, rovny tomu, ve kterém se stroj zastavil. Jinak řečeno, pro každý okamžitý popis s koncovým stavem definujeme triviální instrukci $(q_r, \sigma, q_r, \sigma, 0)$.

Formule $\varphi(x)$ bude obsahovat následující proměnné:

$$\{P^i_{s,t} \mid 1 \le i \le \ell,\, 1 \le s,t \le \tau\},$$
$$\{Q^i_t \mid 1 \le i \le r,\, 1 \le t \le \tau\},$$
$$\{S_{s,t} \mid 1 \le s,t \le \tau\}.$$

Význam proměnných při ohodnocení odpovídajícím přijímajícímu výpočtu je tento

- $P^i_{s,t} = 1$ právě když políčko číslo $s$ obsahuje v čase $t$ symbol $\sigma_i$,
- $Q^i_t = 1$ právě když je stroj v čase $t$ ve stavu $q_i$,
- $S_{s,t} = 1$ právě když v čase $t$ čte hlava políčko $s$,

Formule $\varphi(x)$ má tvar

$$\varphi(x) = \varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_7,$$

kde $\varphi_i$ mají následující podobu a význam.

$\varphi_1$ vyjadřuje, že v každém okamžiku je čteno právě jedno políčko:

$$\varphi_1 = \bigwedge_{t=1}^{\tau} \left( \left( \bigvee_{s=1}^{\tau} S_{s,t} \right) \wedge \bigwedge_{\substack{i,\,j\,=\,1 \\ i\,<\,j}}^{\tau} (\neg S_{i,t} \vee \neg S_{j,t}) \right);$$

$\varphi_2$ vyjadřuje, že v každém okamžiku je stroj v právě jednom stavu:

$$\varphi_2 = \bigwedge_{t=1}^{\tau} \left( \left( \bigvee_{i=1}^{r} Q^i_t \right) \wedge \bigwedge_{\substack{i,\,j\,=\,1 \\ i\,<\,j}}^{r} \left( \neg Q^i_t \vee \neg Q^j_t \right) \right);$$

$\varphi_3$ vyjadřuje, že v každém okamžiku obsahuje každé políčko právě jeden symbol:

$$\varphi_3 = \bigwedge_{s,\,t\,=\,1}^{\tau} \left( \left( \bigvee_{k=1}^{\ell} P^{\ell}_{s,t} \right) \wedge \bigwedge_{\substack{i,\,j\,=\,1 \\ i\,<\,j}}^{\ell} \left( \neg P^i_{s,t} \vee \neg P^j_{s,t} \right) \right);$$

$\varphi_4$ vyjadřuje, že počáteční situace je ve správném tvaru:

$$\varphi_4 = Q^1_1 \wedge S_{1,1} \wedge \bigwedge_{i=1}^{n} P^{k_i}_{i,1} \wedge \bigwedge_{i=n+1}^{\tau} P^1_{i,1}\,,$$

kde $x = \sigma_{k_1}\sigma_{k_2}\cdots\sigma_{k_n}$.

$\varphi_5$ vyjadřuje, že výpočet obsahuje přijímající stav:

$$\bigvee_{t=1}^{\tau} Q^r_t\,.$$

Zbývá zaručit korektnost přechodů. $\varphi_6$ vyjadřuje, že políčko, které není čteno, se nemění:

$$\varphi_6 = \bigwedge_{s,t=1}^{\tau-1} \left( \neg S_{s,t} \Rightarrow \bigwedge_{i=1}^{\ell} \left( P^i_{s,t} \Leftrightarrow P^i_{s,t+1} \right) \right)$$

$\varphi_7$ vyjadřuje, že v každém kroku byla správně použita alespoň jedna instrukce:

$$\varphi_7 = \bigwedge_{t=1}^{\tau-1} \left( \bigvee_{\substack{1 \le s < \tau \\ 1 \le i \le r \\ 1 \le j \le \ell \\ (q_i, \sigma_j, q_{i'}, \sigma_{j'}, m) \\ \in P_M}} \left( Q_t^i \wedge S_{s,t} \wedge P_{s,t}^j \wedge Q_{t+1}^{i'} \wedge S_{s+m,t+1} \wedge P_{s,t+1}^{j'} \right) \right),$$

kde $s' \le \tau$.

Je zřejmé, že uvedenou formuli lze zkonstruovat v čase $\mathcal{O}(p(n)^2)$ a z konstrukce plyne, že má požadovanou vlastnost. Z důvodu přehlednosti jsme použili i jiné spojky než $\wedge$, $\vee$ a $\neg$. Pro úplnost je tedy dodefinujme jako zkratky:

$$\psi_1 \Rightarrow \psi_2 :\equiv \neg\psi_1 \vee \psi_2$$
$$\psi_1 \Leftrightarrow \psi_2 :\equiv (\neg\psi_1 \vee \psi_2) \wedge (\neg\psi_2 \vee \psi_1)$$

□

Vzhledem k tomu, že lze snadno zkonstruovat formuli odpovídající danému booleovskému obvodu, plyne jako okamžitý důsledek **NP**-úplnost problému splnitelnosti booleovských obvodů, označovaný jako CIRCSAT.

**Důsledek 11.2.** CIRCSAT *je* **NP**-*úplný.*

Je jasné, že SAT obsahuje mnoho formulí, jejichž splnitelnost je snadné rozhodnout, např. formule v disjunktivní normální formě. **NP** úplnost tedy plyne z nějaké podmnožiny formulí, které jsou těžké. Ukazuje se, že stačí omezit na formule s poměrně jednoduchým tvarem.

Označme 3SAT množinu booleovských formulí v konjunktivní normální formě, ve kterých navíc každá klausule obsahuje nejvýše tři literály. Vstup si můžeme představit jako seznam nejvýše trojprvkových podmnožin proměnných. Např.

$$\{x_1, x_3\}, \{x_2, \neg x_4, \neg x_5\}, \{\neg x_1, x_2\}$$

representuje formuli

$$(x_1 \vee x_3) \wedge (x_2 \vee \neg x_4 \vee \neg x_5) \wedge (\neg x_1 \vee x_2).$$

**Tvrzení 11.3.** 3SAT *je* **NP**-*úplný.*

*Důkaz.* Redukujeme CIRCSAT na 3SAT. K danému obvodu $C$ se vstupními proměnnými $\{x_1, x_2, \ldots, x_n\}$ a nevstupními vrcholy $V$ zkonstruujeme booleovskou formuli $\varphi_C$, jejíž proměnné budou

$$\{x_1, x_2, \ldots, x_n\} \cup \{x_v \mid v \in V\}.$$

Pro každý vrchol $v$ okruhu $C$ zkonstruujeme formuli $\varphi_v$ v konjunktivní normální formě následovně.

Je-li vrchol $v$ typu $\neg$ se vstupním vrcholem $u$ položme

$$\varphi_v = (x_v \vee x_u) \wedge (\neg x_v \vee \neg x_u).$$

Platí, že ohodnocení $h : \{x_u, x_v\} \to \{0, 1\}$ splňuje $\varphi_v$ právě když $h(x_v) \neq h(\neg x_u)$.

Pro $v$ typu $\wedge$ se vstupními vrcholy $u$ a $w$ položíme

$$\varphi_v = (x_u \vee \neg x_v) \wedge (x_w \vee \neg x_v) \wedge (\neg x_u \vee \neg x_w \vee x_v).$$

Opět platí, že ohodnocení $h : \{x_u, x_v, x_w\} \to \{0, 1\}$ splňuje $\varphi_v$ právě když $h(x_v) = h(x_u \wedge x_w)$.

Nakonec pro $v$ typu $\vee$ se vstupními vrcholy $u$ a $w$ položíme

$$\varphi_v = (\neg x_u \vee x_v) \wedge (\neg x_w \vee x_v) \wedge (x_u \vee x_w \vee \neg x_v),$$

takže $h(\varphi_v) = 1$, právě když $h(x_v) = h(x_u \vee x_w)$.

Označme výstupní vrchol $C$ symbolem $v_F$. Nyní již můžeme definovat

$$\varphi_C = (\bigwedge_{v \in V} \varphi_v) \wedge x_{v_F}\,.$$

Redukce je zřejmě polynomiální. Tvrdíme, že $C$ je splnitelný, právě když $\varphi_C$ je splnitelná.

Nechť je $h$ ohodnocení, které splňuje $\varphi_C$. Pak jistě $h(x_{v_F}) = 1$. Uvažujme ohodnocení vrcholů obvodu $C$ dané ohodnocením $h$ jeho vstupních vrcholů vrcholy. Konstrukce formule zaručuje, že $h(v) = h(x_v)$ pro každé $v \in V$. Tedy i $h(v_F) = 1$ a okruh je ohodnocením $h$ splněn.

Nechť je naopak $h$ ohodnocení vstupních vrcholů obvodu $C$ takové, že $h(v_F) = 1$. Rozšiřme $h$ na proměnné $x_v$, $v \in V$ předpisem $h(x_v) = h(v)$. Z konstrukce formule $\varphi_C$ plyne, že ji ohodnocení $h$ splňuje. □

Další, ještě menší podmnožinou SAT, která je však stále **NP**-úplná je NAESAT (**N**ot **A**ll **E**qual). Je to podmnožina 3SAT obsahující pouze formule, pro něž existuje ohodnocení takové, že pro žádnou klausuli nejsou hodnoty všech literálů stejné. Splnitelnost požaduje, aby hodnota alespoň jednoho literálu v každé klausuli byla 1. NAESAT navíc požaduje, aby hodnota alespoň jednoho literálu byla 0.

Je-li $h$ nějaké ohodnocení proměnných z množiny $X$, pak $\bar{h}$ označíme opačné ohodnocení, tj. takové, pro něž platí $\bar{h}(x) = 1 - h(x)$. Formuli $\varphi \in$ NAESAT můžeme charakterizovat tak, že pro ni existuje ohodnocení $h$ takové, že $h$ i $\bar{h}$ ji splňuje.

**Tvrzení 11.4.** NAESAT *je* **NP**-*úplný.*

*Důkaz.* Provedeme redukci CIRCSAT na NAESAT mírnou úpravou redukce na 3SAT. Nadále budeme používat značení z předchozího důkazu.

Nechť $\varphi'_v$ vznikne z $\varphi_v$ přidáním nové proměnné $z$ do všech klausulí, které obsahují méně než tři literály. Ukážeme, že

$$\varphi'_C = (\bigwedge_{v \in V} \varphi'_v) \wedge (x_{v_F} \vee z)$$

leží v NAESAT, právě když $C$ je splnitelný okruh.

Nechť tedy $h$ a $\bar{h}$ splňuje $\varphi'_C$. Díky symetrii $h$ a $\bar{h}$ můžeme předpokládat $h(z) = 0$. Pak $h$ zjevně splňuje $\varphi_C$, a tedy $C \in$ CIRCSAT.

Nechť naopak $C \in$ CIRCSAT a $h$ splňuje $\varphi_C$. Pak $h$ rozšířené o $h(z) = 0$ splňuje i $\varphi'_C$. Ohodnocení $\bar{h}$ (rozšířené o $\bar{h}(z) = 1$) jistě splňuje klausule obsahující $z$. Zbývá ukázat, že splňuje i klausule, které $z$ neobsahují.

Nechť $v$ je typu $\wedge$ se vstupy $u$ a $w$. Ohodnocení $h$ splňuje $(u \vee \neg v)$ i $(w \vee \neg v)$, odkud snadno odvodíme, že $\bar{h}$ splňuje $(\neg u \vee \neg w \vee v)$.

Nechť je nyní $v$ typu $\vee$ se vstupy $u$ a $w$. Jelikož $h$ splňuje $(\neg u \vee v)$ i $(\neg w \vee v)$, splňuje $\bar{h}$ klausuli $(u \vee w \vee \neg v)$. □

Poslední **NP**-úplný problém, který uvádíme, je jeden z mnoha **NP**-úplných problémů z teorie grafů:

$$\text{3Coloring} = \{G \mid \text{graf } G \text{ je obarvitelný třemi barvami}\}.$$

Připomeňme, že obarvení grafu $G = (V, E)$ třemi barvami je zobrazení $\gamma : V \to \{0, 1, 2\}$ takové, že pokud $(u, v) \in E$, pak $\gamma(u) \neq \gamma(v)$.

**Tvrzení 11.5.** 3Coloring *je* **NP**-*úplný.*

*Důkaz.* Ukážeme redukci NAESAT na 3Coloring. Mějme formuli $\varphi$, která je v konjunktivní normální formě a každá klausule obsahuje dva nebo tři literály. (Pokud nějaká klausule obsahuje jen jeden literál, formule jistě v NAESATneleží.) Tuto vlastnost lze zkontrolovat v polynomiálním čase. (A pokud vstup nemá požadovaný tvar, stačí ho převést na nějaký nepřijatelný vstup 3Coloring.)

Množina proměnných budiž $X = \{x_1, \ldots, x_n\}$ a formule $\varphi$ nechť je konjunkcí $m$ klausulí se třemi literály a $\ell$ klausulí se dvěma literály, tedy tvaru

$$\varphi = \bigwedge_{k=1}^{m} (c_{k,1} \vee c_{k,2} \vee c_{k,3}) \wedge \bigwedge_{k=m+1}^{m+\ell} (c_{k,1} \vee c_{k,2}),$$

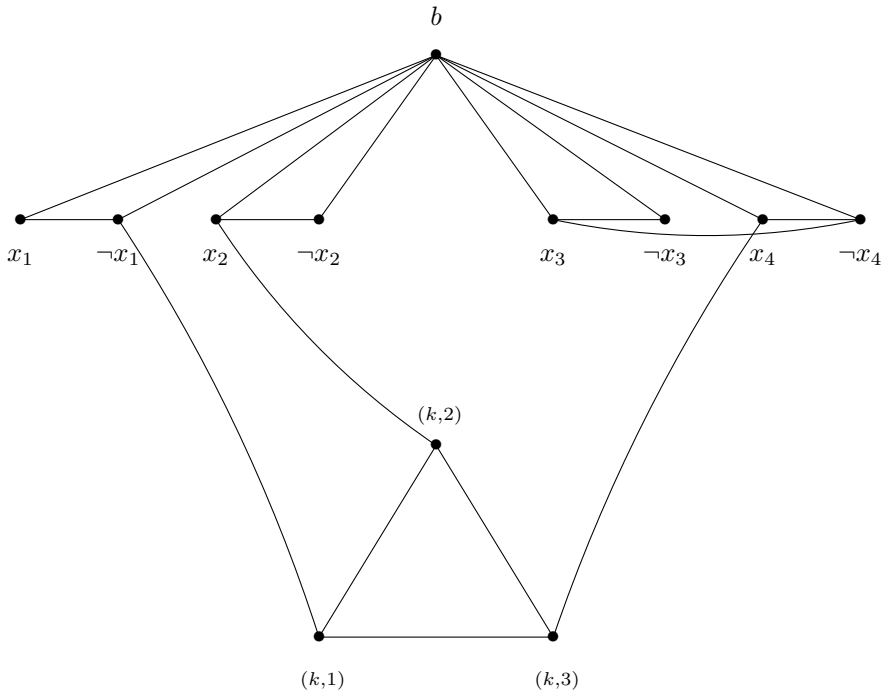
kde $c_{k,i} \in X^{\pm} = \{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$.

Vrcholy $V$ konstruovaného grafu budou sestávat z prvků množiny $X^{\pm}$, uspořádaných dvojic $(k, i)$, $k = 1, \ldots, m$, $i = 1, 2, 3$, a jednoho speciálního vrcholu $b$. Celkem tedy $|V| = 3m + 2n + 1$.

Množina hran $E$ je následující:

(1) $(x_i, \neg x_i) \in E$ pro všechna $i = 1, \ldots, n$;
(2) $(x_i, b) \in E$, $(\neg x_i, b) \in E$, pro všechna $i = 1, \ldots, n$;
(3) $((k, i), (k, j)) \in E$ pro všechna $k = 1, 2, \ldots, m$ a $i, j = 1, 2, 3$, $i \neq j$;
(4) $((k, i), y) \in E$ pro všechna $(k, i)$ a $y \in X^{\pm}$ taková, že $y = c_{k,i}$;
(5) $(c_{k,1}, c_{k,2})$ pro všechna $k = m + 1, m + 2, \ldots, m + \ell$.

Následující obrázek ilustruje část grafu odpovídající $k$-té klausuli $(\neg x_1 \vee x_2 \vee x_4)$ a klausuli $(x_3 \vee \neg x_4)$.




Tím je popis grafu $G = (V, E)$ dokončen. Je zřejmé, že redukce je polynomiální. Ukážeme, že $G \in$ 3Coloring, právě když $\varphi \in$ NAESAT.

Nechť je nejprve $G \in$ 3Coloring a $\gamma : V \to \{0, 1, 2\}$ je jeho obarvení třemi barvami. Můžeme předpokládat $\gamma(b) = 2$. Odtud plyne, díky hranám uvedeným pod body (1) a (2), že $\gamma(x) \in \{0, 1\}$ a $\gamma(x) \neq \gamma(\neg x)$ pro všechna $x \in X$. Můžeme tedy definovat ohodnocení $h$ množiny $X$ předpisem $h(x) = \gamma(x)$. Ukážeme, že $h$ splňuje $\varphi$. Zvolme $k \in \{1, \ldots, m\}$. Vzhledem k tomu, že vrcholy $(k, i)$, $i = 1, 2, 3$, jsou

spojeny hranami, existuje $i_0$ takové, že $\gamma(k, i_0) = 0$. Kvůli hranám popsaným v bodě (4) je $\gamma(c_{k,i_0}) = 1$, a tudíž také $h(c_{k,i_0}) = 1$. Pro $k = m+1, m+2, \ldots, m+\ell$ platí díky hraně z bodu (5), že $h(c_{k,i}) = 1$ pro právě jedno $i = 1, 2$. Ohodnocení $h$ tedy splňuje všechny klausule. Stejná úvaha je možná i pro $\bar{h}$, a proto $\varphi \in$ NAESAT.

Nechť je nyní $\varphi \in$ NAESAT a $h$ a $\bar{h}$ jsou opačná ohodnocení, která splňují $\varphi$. Vrchol $b$ a vrcholy $X^{\pm}$ obarvíme takto

- $\gamma(b) = 2$;
- $\gamma(c_{k,i}) = h(c_{k,i})$, pro všechna $c_{k,i} \in X^{\pm}$.

Uvažujme nyní klausuli $(c_{k,1} \vee c_{k,2} \vee c_{k,3})$. Vzhledem k tomu, že $h$ je ohodnocení podle podmínek NAESAT, existují $i_1, i_2 \in \{1, 2, 3\}$ taková, že $h(c_{k,i_1}) \neq h(c_{k,i_2})$. Definujme

- $\gamma((k, i_1)) = \bar{h}(c_{k,i_1})$;
- $\gamma((k, i_2)) = \bar{h}(c_{k,i_2})$;
- $\gamma((k, i_3)) = 2$, kde $\{i_1, i_2, i_3\} = \{1, 2, 3\}$.

Přímočaře ověříme, že $\gamma$ je obarvení $G$. □

Zmiňme na závěr bez důkazů (které nejsou těžké) dvě varianty problému SAT, které leží v **P**. Klausule se nazývá *Hornova*, pokud obsahuje nejvýše jeden pozitivní literál. Definujme

$$\text{HornSAT} = \{\varphi \mid \varphi \text{ je splnitelná konjunkce Hornových klauzulí}\}.$$

**Tvrzení 11.6.** HornSAT $\in$ **P**.

Snížíme-li počet literálů v klausuli ze tří na dva, stane se problém řešitelný v polynomiálním čase.

$$\text{2SAT} = \{\varphi \mid \varphi \in \textbf{CNF} \text{ a každá klausule obsahuje nejvýše dva literály}\}.$$

**Tvrzení 11.7.** 2SAT $\in$ **P**.

## 12. Pravděpodobnostní odhady

Důležitým aspektem pravděpodobnostních algoritmů, kterými se budeme zabývat dále, je možnost opakovat stejný výpočet vícekrát. Zatímco pro deterministický algoritmus nedává takový postup žádný smysl, u pravděpodobnostních algoritmů se opakováním mohou dramaticky měnit pravděpodobnosti správné odpovědi. Má-li např. nějaký polynomiální algoritmus pravděpodobnost úspěchu $1/n^k$, což bychom intuitivně považovali za pravděpodobnost příliš malou, stačí opakovat tento algoritmus $n^{k+\ell}$-krát a snížíme pravděpodobnost neúspěchu na

$$\left(1 - \frac{n^\ell}{n^{k+\ell}}\right)^{n^{k+\ell}} \longrightarrow e^{-n^\ell}.$$

Polynomiálně malou úspěšnost je tedy možné změnit na exponenciálně malou neúspěnost, přičemž celková složitost zůstane polynomiální. Tato úvaha platí pro případ, kdy úspěšný běh algoritmu s jistotou poznáme, např. pokud algoritmus generuje svědka pro nějaký jazyk v **NP**.

Důležité jsou ale také situace, kdy správnou odpověď nepoznáme a rozhodujeme se na základě většinového výsledku. V takových případech potřebujeme pravděpodobnostní odhady související se zákonem velkých čísel: průměrná hodnota opakovaného experimentu se blíží jeho střední hodnotě. Ale jak rychle a s jakou pravděpodobností? Na to odpovídají následující odhady. Důležitou okolností je míra nezávislosti opakovaných experimentů:

- Markovova nerovnost platí pro jakoukoli náhodnou veličinu, ale je nejslabší.
- Silnější Čebyševova nerovnost platí pro *po dvou* nezávislé veličiny.
- Nejsilnější je Chernoffův odhad, který ale vyžaduje zcela nezávislé veličiny.

Zopakujme nejprve základní pojmy teorie pravděpodobnosti. Množinu $\Omega$ nazveme *pravděpodobnostním prostorem* pokud je na ní definována míra $\mu$ taková, že $\mu(\Omega) = 1$. Pro naše potřeby stačí uvažovat spočetné pravděpodobnostní prostory. Typicky bude $\Omega = \mathbb{N}$ nebo $\Omega = \{0,1\}^*$. To umožňuje psát místo integrálů sumy.

Náhodná veličina $X$ je zobrazení $X : \Omega \to \mathbb{R}$. *Střední* (neboli *očekávaná*) *hodnota* $X$ je

$$\mathrm{E}(X) = \sum_{\omega \in \Omega} X(\omega)\mu(\omega).$$

Pravděpodobnost nějakého jevu $A$ budeme značit $\Pr[A]$. Je dána mírou podmnožiny pravděpodobnostního prostoru, pro kterou jev nastává.

Výše jsme popsali situaci, kdy zkoumáme náhodnou veličinu $X = \sum_{i=1}^{n} Z_i$. Pro její střední hodnotu platí

$$\mathrm{E}(X) = \sum_{i=1}^{n} \mathrm{E}(Z_i),$$

a to bez ohledu na závislost či nezávislost veličin $Z_i$. Klíčová otázka zní, jaká je pravděpodobnost, že se hodnota veličiny $X$ bude od očekávané hodnoty daným způsobem lišit.

Dokážme první odhad.

**Věta 12.1** (Markovova nerovnost)**.** *Nechť je $X$ náhodná veličina s kladnými hodnotami a $k > 0$. Pak*

$$\Pr[X \geq k] \leq \frac{\mathrm{E}(X)}{k}.$$

*Důkaz.*

$$\begin{aligned}\mathrm{E}(X) = \sum_{\omega \in \Omega} X(\omega)\mu(\omega) &= \sum_{X(\omega)<k} X(\omega)\mu(\omega) + \sum_{X(\omega)\geq k} X(\omega)\mu(\omega) \\ &\geq \sum_{X(\omega)\geq k} X(\omega)\mu(\omega) \geq k \cdot \sum_{X(\omega)\geq k} \mu(\omega) = k \cdot \Pr[X \geq k].\end{aligned}$$

□

Jak moc se náhodná veličina pro jednotlivé prvky pravděpodobnostního prostoru liší od své očekávané hodnoty vyjadřuje *rozptyl* neboli *variance*, definovaná

$$\mathrm{Var}(X) := \mathrm{E}((X - \mathrm{E}(X))^2) = \mathrm{E}(X^2) - \mathrm{E}(X)^2 \ .$$

Jednoduchou aplikací Markovovy nerovnosti na formuli rozptylu dostáváme druhý odhad.

**Věta 12.2** (Čebyševův odhad)**.**

$$\Pr[\,|X - \mathrm{E}(X)| \geq t\,] \leq \frac{\mathrm{Var}(X)}{t^2} \ .$$

*Důkaz.*

$$\Pr[\,|X - \mathrm{E}(X)| \geq t\,] = \Pr\left[\,(X - \mathrm{E}(X))^2 \geq t^2\,\right] \overset{\text{Markov}}{\leq} \frac{\mathrm{Var}(X)}{t^2} \ .$$

□

Pro aplikaci Čebyševovy nerovnosti pro případ $X = \sum Z_i$ je zásadní skutečnost, že pokud jsou veličiny $Z_i$ po dvou nezávislé, platí

$$\mathrm{Var}\left(\sum Z_i\right) = \sum \mathrm{Var}(Z_i).$$

Jsou-li $Z_i$ po dvou nezávislé kopie náhodné veličiny $Z$, dává Čebyševova nerovnost pro $X = \sum_{i=1}^{m} Z_i$ odhad

$$\Pr\left[\left|\frac{X}{m} - \mathrm{E}(Z)\right| \geq \varepsilon\right] \leq \frac{\mathrm{Var}(Z)}{m\,\varepsilon^2}\ .$$

*Příklad* 12.3. Typickou zkoumanou situací je opakování experimentu s Bernoulliho rozdělením se střední hodnotou $0 \leq p \leq 1$, tedy náhodné veličiny $Z$ nabývající hodnot $\{0, 1\}$ takové, že

$$\Pr[Z = 1] = p \qquad\qquad \Pr[Z = 0] = 1 - p\,.$$

Snadno ověříme, že $E(Z) = p$ a $\mathrm{Var}(Z) = p(1-p)$. Jsou-li $Z_i$ po dvou nezávislé kopie $Z$, dostáváme pro $X = \sum_{i=1}^{m} Z_i$ odhad

$$\Pr\left[\left|\frac{X}{m} - p\right| \geq \varepsilon\right] \leq \frac{p\,(1-p)}{m\,\varepsilon^2}\ .$$

Jsou-li sčítané veličiny zcela nezávislé, můžeme odhad výrazně zlepšit Chernoffovým odhadem. Odhad využívá fakt, že střední hodnota součinu nezávislých veličin je součin jejich středních hodnot. Chernoffův odhad ukážeme v přesné formulaci pro součet náhodných veličin se stejným Bernoulliho rozdělením.

**Věta 12.4** (Chernoffův odhad)**.** *Uvažujme nezávislé kopie náhodných veličin $Z_i$, $i = 1, \ldots, m$ náhodné veličiny $Z$ s Bernoulliho rozdělením a střední hodnotou $p$ a položme*

$$X = \sum_{i=1}^{m} Z_i.$$

*Pak platí*

$$\Pr\left[\frac{X}{m} \geq p + \varepsilon\right] \leq \exp\left(-m \cdot \mathrm{D}(p + \varepsilon \parallel p)\right) \tag{12.1}$$

$$\Pr\left[\frac{X}{m} \leq p - \varepsilon\right] \leq \exp\left(-m \cdot \mathrm{D}(1 - p + \varepsilon \parallel 1 - p)\right), \tag{12.2}$$

*kde*

$$\mathrm{D}(x \parallel y) := x \ln\left(\frac{x}{y}\right) + (1-x)\ln\left(\frac{1-x}{1-y}\right).$$

*Důkaz.* Pro libovolné $t > 0$ a libovolné $k > 0$ platí

$$\begin{aligned}
\Pr[X \geq k] &= \Pr\left[e^{tX} \geq e^{tk}\right] \\
&\leq \frac{\mathrm{E}\left(e^{tX}\right)}{e^{tk}} && \text{podle Markovovy nerovnosti,} \\
&= \frac{\left(\mathrm{E}\left(e^{tZ_i}\right)\right)^m}{e^{tk}} && \text{z nezávislosti } Z_i, \\
&= \frac{(p\,e^t + 1 - p)^m}{e^{tk}} && \text{přímým výpočtem } \mathrm{E}(\exp(tZ_i)).
\end{aligned}$$

Pro

$$t = \ln\frac{(1-p)\,(p+\varepsilon)}{p\,(1-p-\varepsilon)} \qquad \text{a} \qquad k = m\,(p+\varepsilon)$$

dostáváme vztah (12.1). Nerovnost (12.2) je ekvivalentní nerovnosti (12.1) pro veličinu $Y = \sum_{i=1}^{m}(1 - Z_i)$. □

Funkce $\mathrm{D}(x \parallel y)$ je *relativní entropie* Bernoulliho distribucí se střední hodnotou $x$ a $y$ (všimněte si, že záleží na pořadí, formule není symetrická). Z konkávnosti logaritmu dostáváme

$$-\mathrm{D}(x \parallel y) = x \ln\left(\frac{y}{x}\right) + (1-x)\ln\left(\frac{1-y}{1-x}\right) \leq \ln 1 = 0\,.$$

Relativní entropie je tedy vždy kladná pro $x \neq y$ (a nulová pro $x = y$) a Chernoffův odhad poskytuje exponenciálně rychle klesající pravděpodobnost odchylky od očekávané hodnoty s rostoucímpočtem opakování.

Analýzou relativní entropie lze získat následující praktičtější odhady, které uvádíme bez důkazu. Z odhadu $\mathrm{D}(p+\varepsilon \parallel p) \geq 2\varepsilon^2$ plyne

$$\Pr\left[\left|\frac{X}{m} - p\right| \geq \varepsilon\right] \leq 2\exp\left(-2\,m\,\varepsilon^2\right)\,.$$

Často se uvádějí multiplikativní verze Chernoffova odhadu:

$$\Pr\left[\frac{X}{m} \leq (1-\delta)\,p\right] \leq \exp\left(-\,m\,p\cdot\frac{\delta^2}{2}\right)\,.$$

$$\Pr\left[\frac{X}{m} \geq (1+\delta)\,p\right] \leq \exp\left(-\,m\,p\cdot\frac{\delta^2}{2+\delta}\right)\,.$$

## 13. Pravděpodobnostní algoritmy a složitostní třídy

Nedeterministický Turingův stroj je výpočetní model, jehož definice působí velmi nerealisticky, protože předpokládáme, že všechny „větve“ výpočtu probíhají současně. V reálném situaci je přirozené předpokládat, že stroj si musí jedno z pokračování vybrat. Pokud stanovíme s jakou pravděpodobností si jednotlivé definované možnosti vybírá, dostáváme definici pravděpodobnostního Turingova stroje.

**Definice 13.1.** Pravděpodobnostní Turingův stroj je nedeterministický Turingův stroj spolu s funkcí $P : (Q \times A \times Q \times A \times M) \to [0,1]$ takovou, že pro každé $(q,a) \in Q \times A$ platí

$$\sum_{(q',a',m')\in Q\times A\times M} P(q,a,q',a',m) = 1$$

kde $Q$ je množina stavů, $A$ je množina symbolů a $M = (\leftarrow, \rightarrow, -)$ množina pohybů.

Třídu pravděpodobnostních Turingových strojů budeme značit **PTM**. Funkce $P$ určuje s jakou pravděpodobností použije stroj v okamžitém stavu $(q,a)$ instrukci $(q,a,q',a',m')$. Funkce je definovaná pro všechny možné instrukce. Případ $P(q,a,q',a',m) = 0$ odpovídá situaci, v níž se daná instrukce v programu nevyskytuje (je použita s nulovou pravděpodobností).

Časová i prostorová náročnost pravděpodobnostních strojů se definuje podobně jako u strojů nedeterministických, tedy spotřebou v nejnáročnějším případě.

Pravděpodobnostní stroje hrají v kryptografii klíčovou úlohu, protože bezpečnost systému je ohrožena nejen tehdy, pokud existuje útok, který je úspěšný vždy, ale také tehdy, pokud je útok úspěšný s nezanedbatelnou pravděpodobností.

Pro jednoduchost budeme často pracovat s *normalizovanými pravděpodobnostními stroji*, u kterých jsou pro všechny okamžité popisy právě dvě instrukce s pravděpodobností $\frac{1}{2}$. Každý pravděpodobnostní stroj lze simulovat normalizovaným strojem s vysokou přesností a konstantní ztrátou efektivity. Stačí rozdělit interval $[0,1]$ na podintervaly s délkami rovnými pravděpodobnostem jednotlivých možných instrukcí a poté postupnou uniformní volbou bitů $b_i$ konstruovat číslo $r = 0, b_1 b_2 \ldots$, dokud nedosáhneme přesnosti umožňující rozhodnout, v jakém intervalu se $r$ nachází. Podobně jako v případě vektoru voleb u nedeterministického

stroje lze takto pravděpodobností část výpočtu koncentrovat na začátek, kdy dojde k vygenerování potřebného počtu náhodných bitů, a poté je již výpočet deterministický.

Uvažujme nějaký výpočet stroje $M \in \mathbf{PTM}$, který sestává z aplikace posloupnosti instrukcí $i_1, i_2, \ldots, i_t$. Pravděpodobnost takového výpočtu je

$$\prod_{k=1}^{t} P(i_k).$$

Řekneme, že $M \in \mathbf{PTM}$ *přijímá vstup $x$ s pravděpodobností $p$*, pokud součet pravděpodobností přijímajících výpočtů je $p$.

**Definice 13.2.** $L \in \mathbf{RP}$ právě když existuje $M \in \mathbf{PTM}$ pracující v polynomiálním čase takový, že

a) pokud $x \notin L$, pak $M$ odmítne (vždy),
b) pokud $x \in L$, pak $M$ přijme s pravděpodobností alespoň $1/2$.

Stroje, které přimají jazyky **RP** (zkratka pro „**r**andomized **p**olynomial time“) nazýváme stroje typu MONTE CARLO. Platí pro ně, že v nich nedochází k chybnému přijímání (no false positives), ale odmítnutí chybné být může.

Snadno si rozmyslíme, že platí

**Tvrzení 13.3.** $\mathbf{P} \subseteq \mathbf{RP} \subseteq \mathbf{NP}$

Následující tvrzení ukazuje, že hodnota $\frac{1}{2}$ v definici **RP** je nepodstatná a může být nahrazena libovolnou hodnotou, polynomiálně malou.

**Věta 13.4.** *Jazyk $L$ leží v* **RP***, právě když existuje $M \in \mathbf{PTM}$ pracující v polynomiálnám čase a $k \in \mathbb{N}$ takové, že*

a) *je-li $x \notin L$ odmítne,*
b) *je-li $x \in L$ přijme s pravděpodobností alespoň $\frac{1}{|x|^k}$.*

*Důkaz.* Přímá implikace je zřejmá.

Mějme nyní stroj $M$ vyhovující podmínkám věty. Chceme ukázat, že existuje stroj $M'$ odpovídající definici. Takový stroj spočívá v opakování běhu stroje $M$. Počet opakování bude $n^k$, kde $n = |x|$. Stroj $M'$ přijme právě tehdy, pokud $M$ při opakovaných pokusech přijme alespoň jednou.

Vzhledem k tomu, že $M$ nemá žádná chybná přijetí, nemá chybná přijetí ani $M'$. Spočítejme pravděpodobnost chybného odmítnutí. Pravděpodobnost chybného odmítnutí v každém kole běhu $M$ je $(1 - \frac{1}{n^k})$. Předpokládáme, že jednotlivá kola jsou nezávislá, a proto celková pravděpodobnost chyby je nejvýše

$$\left(1 - \frac{1}{n^k}\right)^{n^k} < \frac{1}{2}.$$

□

Podobná úvaha umožňuje pravděpodobnost omylu exponenciálně snížit, jak ukazuje následující tvrzení.

**Věta 13.5.** *Jazyk $L$ leží v* **RP***, právě když existuje $M \in \mathbf{PTM}$ pracující v polynomiálnám čase a $k \in \mathbb{N}$ takové, že*

a) *je-li $x \notin L$ odmítne,*
b) *je-li $x \in L$ přijme s pravděpodobností alespoň $1 - \frac{1}{2^{|x|}}$.*

*Důkaz.* Zde je zřejmá implikace opačná.

Máme-li stroj s pravděpodobností omylu menší než jedna polovina, opakujeme jeho běh $n$-krát, čímž snížíme pravděpodobnost omylu pod $\left(\frac{1}{2}\right)^n$. □

Označením **co-RP**, budeme jako obvykle mínit třídu jazyků, jejichž komplement je v **RP**. Jazyk je tedy v **co-RP** pokud existuje stroj, který se nedopouští chybného odmítnutí, a pravděpodobnost chybného přijetí je menší než $\frac{1}{2}$.

Průnik obou tříd definuje třídu

$$\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{co\text{-}RP}.$$

Stroje přijímající jazyky třídy **ZPP** (**z**ero error **p**robability in **p**olynomial time) se nazývají algoritmy LAS VEGAS a můžeme si je představit jako současně běžící dva stroje typu MONTE CARLO, jeden s jistým přijetím, druhý s jistým odmítnutím.

S pravděpodobností alespoň jedna polovina se po běhu algoritmu dozvíme výsledek. To tehdy, pokud jeden ze strojů poskytne odpověď, ve které se nemýlí. Název třídy je odvozen z faktu, že opakováním běhu algoritmu polynomiálně mnohokrát se s exponenciální jistotou dozvíme správnou odpověď. Z jiného pohledu, pokud necháme stroj běžet tak dlouho, dokud nedá jistou odpověď, bude průměrná doba čekání polynomiální. (Přesněji, průměrná doba čekání bude dvě kola.)

U tříd **RP** a **ZPP** je nápadná asymetrie kladného a záporné odpovědi. Taková asymetrie je v některých případech přirozená. Např. pokud se snažíme ověřit prvočíselnost nějakého čísla tak, že ho zkusíme vydělit nějakým menším číslem. Pokud uspějeme, je zřejmé, že číslo prvočíslem není. V opačném případě se můžeme domnívat, že prvočíslem je, ale takový výsledek je vystaven omylu (v tomto případě samozřejmě mnohem většímu, než $\frac{1}{2}$). Např. Rabinův-Millerův test už ale ukazuje, že prvočísla leží v **co-RP**.

Třídu, ve které je omyl rovnoměrně rozložen do kladných i záporných odpovědí, definujeme nyní.

**Definice 13.6.** $L \in \mathbf{BPP}$ právě když existuje $M \in \mathbf{PTM}$ pracující v polynomiálním čase takový, že

a) je-li $x \in L$, pak pravděpodobnost přijetí je alespoň $\frac{2}{3}$,
b) je-li $x \notin L$, pak pravděpodobnost přijetí je nejvýše $\frac{1}{3}$.

Ani v případě této definice není hodnota $\frac{2}{3}$ podstatná. Pravděpodobnost správné odpovědi je opět možné zvýšit opakováním běhu stroje. V tomto případě ale nelze čekat na jistě správnou odpověď. Musíme se spokojit s většinovou odpovědí, viz diskusi v kapitole o pravděpodobnostních odhadech. Nemusí ale jít o prostou většinu. Důležité je, že stroj se chová jinak pro vstupy z jazyka v porovnání s chováním pro vstupy mimo jazyk. Konkrétně stačí, že pravděpodobnost odpovědi „ano“ je pro vstupy z jazyka o poznání pravděpodobnější než pro vstupy mimo jazyk. Význam výrazu „o poznání“ je dán Chebyshevovým odhadem, jak ukazuje následující věta a její důkaz.

**Věta 13.7.** *Následující podmínky jsou ekvivalentní.*

1. $L \in \mathbf{BPP}$
2. *Existuje* $M \in \mathbf{PTM}$ *pracující v polynomiálním čase,* $c \in (0,1)$ *a* $k \in \mathbb{N}$ *takové, že*
   a) *je-li* $x \in L$, *pak pravděpodobnost přijetí je alespoň* $c + \frac{1}{|x|^k}$,
   b) *je-li* $x \notin L$, *pak pravděpodobnost přijetí je nejvýše* $c - \frac{1}{|x|^k}$.
3. *Pro každé* $\ell \in \mathbb{N}$ *existuje* $M \in \mathbf{PTM}$ *pracující v polynomiálním čase takový, že*
   a) *je-li* $x \in L$, *pak pravděpodobnost přijetí je alespoň* $1 - \frac{1}{2^{|x|^\ell}}$,
   b) *je-li* $x \notin L$, *pak pravděpodobnost přijetí je nejvýše* $\frac{1}{2^{|x|^\ell}}$.

*Důkaz.* Stačí ukázat, že z 2. plyne 3. Ostatní implikace jsou zřejmé.

Nechť stroj $M \in \mathbf{PTM}$ splňuje předpoklady tvrzení 2. Zkonstruujeme stroj $M'$ odpovídající pro dané $\ell$ požadavkům tvrzení 3. Práce stroje $M'$ spočívá v $|x|^{2k+\ell}$ opakování běhu stroje $M$. $M'$ přijme vstup $x$ právě tehdy, když $M$ přijal $x$ ve více než $c|x|^{2k+\ell}$ kolech.

Spočtěme, jaká je pravděpodobnost chyby algoritmu $M'$. Předpokládejme, že $x$ leží v $L$, opačný případ je analogický. Označme $Z$ náhodnou veličinu definovanou takto:

$$Z = \begin{cases} 1, & \text{pokud } M \text{ přijal } x; \\ 0, & \text{pokud } M \text{ nepřijal } x. \end{cases}$$

Označme $X$ náhodnou veličinu, která je součtem $t(n)$ kopií veličiny $Z$. $X$ je tedy počet kol, ve kterých $M$ odpověděl správně.

Střední hodnota veličiny $X/|x|^{2k+\ell}$, označme ji $p$, je přitom stejná jako střední hodnota $Z$, tedy $p \geq c + 1/|x|^k$. Stroj $M'$ odmítne $x$ právě tehdy, když $X/|x|^{2k+\ell} \leq c$. Jednotlivé běhy stroje $M$ považujeme za nezávislé a proto můžeme použít Chernoffův odhad. Dostáváme

$$\Pr\left[\frac{X}{|x|^{2k+\ell}} \leq c\right] \leq \Pr\left[\left|\frac{X}{|x|^{2k+\ell}} - p\right| \geq \frac{1}{|x|^k}\right] \leq 2\exp\left(-2\,\frac{|x|^{2k+\ell}}{|x|^{2k}}\right) \leq \frac{1}{2^{|x|^\ell}}\,.$$

Pravděpodobnost chyby algoritmu $M'$ je tedy menší než $2^{-|x|^\ell}$, což jsme měli dokázat. □

Předchozí větu můžeme shrnout slovy, že pokud umíme nějaký jazyk rozhodovat v polynomiálním čase s pravděpodobností úspěchu, který je alespoň o převrácenou hodnotu polynomu lepší než obyčejné hádání, pak ho v polynomiálním čase umíme rozhodovat s pravděpodobností exponenciálně blízkou jedné.

## 14. Neuniformní složitost

V této kapitole se budeme zabývat velkorysou složitostní třídou, která slouží spíše jako horní odhad realistických výpočtů. Tato třída do jisté míry porušuje základní vlastnost algoritmu, totiž jeho uniformitu. Na vstupy různé délky používáme odlišné algoritmy.

**Definice 14.1.** $L \in \mathbf{P/poly}$ právě když existuje posloupnost Booleovských okruhů $\{C_i\}_{i=1}^{\infty}$ a polynom $p$ takové, že $C_i$ má právě $i$ vstupů, $|C_i| \leq p(i)$ a pro každé $x$, $|x| = i$, $C_i(x) = 1$, právě když $x \in L$.

Ekvivalentní charakteristika třídy $\mathbf{P/poly}$ je následující.

**Tvrzení 14.2.** *$L \in \mathbf{P/poly}$, právě když existuje polynomiální $T \in \mathbf{DTM}$, posloupnost $\{a_n\}_{n=1}^{\infty}$ a polynom $p$ takové, že pro všechna $n \in \mathbb{N}$ je $|a_n| \leq p(n)$ a pro každé $x$, $|x| = n$, platí, že $x \in L$, právě když $M(x, a_n)$ přijímá.*

*Důkaz.* Je-li $L \in \mathbf{P/poly}$, pak lze za $a_n$ zvolit nějaký kód $C_n$ a $M$ bude zjišťovat jeho výstup.

Pokud naopak $L$ splňuje podmínky věty, vytvoříme obvod $C_n$ podobně jako v důkazu Cookovy-Levinovy věty tak, aby simuloval výpočet stroje $M$ s konstantním dodatečným vstupem $a_n$. □

Jak jsme již poznamenali, neuniformní třída je založena na používání nekonečně mnoha různých algoritmů. Buď nekonečně mnoho obvodů nebo jeden algoritmus s nekonečně mnoha nápovědami $a_n$.

Jsou jen dvě omezení:

- pro vstupy stejné délky se používá stejný obvod (stejná nápověda)
- obvody (nápověda) mají polynomiálně omezenou velikost.

Zřejmě

$$\mathbf{P} \subset \mathbf{P/poly},$$

stačí uvažovat prázdnou nápovědu.

Nápověda umožňuje stroji rychle určit, jaká slova dané délky v jazyce leží. Pokud je takových slov málo, může jako nápověda sloužit jejich seznam. Řekneme, že jazyk $L$ je *řídký* (ang. *sparse*), pokud existuje polynom $q$ takový, že pro každé $n$ je počet slov délky $n$ v $L$ nejvýše $q(n)$. Z výše uvedené úvahy plyne, že $\mathbf{P/poly}$ obsahuje všechny řídké jazyky. Z toho plyne následující fakt.

**Tvrzení 14.3.** *Existuje nerekursivní jazyk $L \in \mathbf{P/poly}$.*

*Důkaz.* Stačí ukázat, že existují nerekursivní řídké jazyky. Uvažme např. unární jazyk $K = \{1^{G(x)} \mid x \in L\}$, kde $L$ je nerekursivní jazyk nad $\Sigma$ a $G$ je efektivní očíslování $\Sigma^+$. □

Posloupnost okruhů $\{C_n\}_{n=1}^{\infty}$, pro kterou existuje polynomiální $M \in \mathbf{DTM}$, který na vstupu $1^n$ vygeneruje kód $C_n$ se nazývá *uniformní*. Třídu $\mathbf{P}$ lze chápat jako třídu jazyků, pro které existuje uniformní posloupnost okruhů. Totéž by platilo pro uniformní nápovědy. Třída je tedy silná díky nápovědám, které jsou sice polynomiálně dlouhé, ale nejsou generovatelné v polynomiálním čase. O existenci takové obtížně nalezitelné nápovědy mluví následující tvrzení.

**Tvrzení 14.4.** $\mathbf{BPP} \subseteq \mathbf{P/poly}$

*Důkaz.* Nechť $L \in \mathbf{BPP}$. Pak podle Věty 10.9. existuje polynomiální $M \in \mathbf{PTM}$ rozhodující $L$ s pravděpodobností chyby menší než $2^{-|x|}$. Předpokládejme, že stroj nejprve zvolí vektor náhodných bitů $r$, a poté provede deterministický výpočet $M(x, r)$. (Zde, přísně vzato, předpokládáme, že stroj je normalizovaný.)

Chceme ukázat (nekonstruktivně), že pro každé $n$ existuje alespoň jeden vektor voleb stroje $M$, který se nedopouští chyby pro žádný vstup délky $n$. Ten potom zvolíme jako nápovědu $a_n$. Vznikne tak neuniformní stroj, který na vstupu $x$ simuluje běh stroje $M$ s příslušným vektorem náhodnosti $a_{|x|}$.

Označme

$$V_x = \{r \mid M(r, x) \neq \chi_L(x)\}.$$

Buď $R$ náhodná veličina, jejíž hodnotou je vygenerovaný vektor náhodnosti $r$. Pro libovolné $x$ platí

$$\Pr[R \in V_x] < 2^{-|x|},$$

a tedy

$$\Pr\left[R \in \bigcup_{|x|=n} V_x\right] \leq \sum_{|x|=n} \Pr[R \in V_x] < 2^n \cdot 2^{-n} = 1.$$

Pravděpodobnost, že při náhodné volbě zvolíme vektor náhodnosti, který se mýlí pro alespoň jeden vstup, je menší než 1. Existuje tedy volba, pro kterou takový vstup neexistuje. To jsme chtěli ukázat. □

## 15. Rozlišitelnost náhodných distribucí

Ve větách amplifikujících pravděpodobnost jsme viděli, že libobolný polynomiální algoritmus, jehož pravděpodobnost úspěchu je alespoň převrácená hodnota polynomu, umožňuje snížit pravděpodobnost chyby exponenciálně. To motivuje následující definici funkce, kterou, pokud vyjadřuje např. pravděpodobnost úspěchu algoritmu, lze ještě považovat za zanedbatelnou.

**Definice 15.1.** $\mu : \mathbb{N} \to \mathbb{R}_+$ je *zanedbatelná* funkce, pokud pro každý polynom $p$ existuje $n_p \in \mathbb{N}$ takové, že

$$\mu(n) < \frac{1}{p(n)}$$

pro všechna $n \geq n_p$.

Při výzkumu algoritmických problémů se obvykle snažíme nalézt efektivní algoritmus a jeho neexistenci chápeme jako negativní výsledek. Situace je jiná v případě pseudonáhodných generátorů. Zde je naopak nemožnost rozeznat pseudonáhodný výstup od skutečně náhodné posloupnosti žádoucí.

Dále budeme pracovat se soubory náhodných veličin $\{X_n\}_{n\in\mathbb{N}}$, kde $X_n$ nabývá hodnot z $\{0,1\}^{\ell(n)}$, kde $\ell : \mathbb{N} \to \mathbb{N}$ je nějaké zobrazení, typicky splňující $\ell(n) \geq n$ (většinou budeme uvažovat $\ell(n) = n$). Zejména nás bude zajímat, kdy je takový soubor blízko souboru příslušných rovnoměrně rozdělených náhodných veličin $\{U_{\ell(n)}\}_{n\in\mathbb{N}}$.

Blízkost souborů může mít různé stupně.

**Definice 15.2.** Nechť $\{X_n\}_{n\in\mathbb{N}}$ a $\{Y_n\}_{n\in\mathbb{N}}$ jsou soubory náhodných veličin.

- Soubory jsou *identické*, pokud pro všechna $n \in \mathbb{N}$ platí $X_n = Y_n$, tedy veličiny $X_n$ a $Y_n$ mají stejné rozdělení.
- Soubory jsou *statisticky blízké*, pokud je jejich *statistická diference*, definovaná jako

  $$\Delta_{X,Y}(n) := \frac{1}{2} \sum_{v\in\{0,1\}^n} \left|\Pr\left[X_n = v\right] - \Pr\left[Y_n = v\right]\right|,$$

  zanedbatelná funkce.
- Soubory jsou *výpočetně nerozlišitelné*, pokud pro každý polynomiální pravděpodobnostní algoritmus $D$ je

  $$\delta_{D,X,Y}(n) := \left|\Pr\left[D(X_n) = 1\right] - \Pr\left[D(Y_n) = 1\right]\right|$$

  zanedbatelná funkce. (Pravděpodobnost bereme přes rozdělení náhodných veličin $X_n$ a $Y_n$ i přes náhodnost běhu algoritmu $D$.)
- Soubory jsou *silně výpočetně nerozlišitelné*, pokud pro každý soubor obvodů $\mathcal{C} = \{C_n\}_{n\in\mathbb{N}}$ polynomiální velikosti platí

  $$\sigma_{\mathcal{C},X,Y}(n) := \left|\Pr\left[C_n(X_n) = 1\right] - \Pr\left[C_n(Y_n) = 1\right]\right|$$

  zanedbatelná funkce.

Definice pseudonáhodnosti se opírá o pojem (silné) výpočetní nerozlišitelnosti.

**Definice 15.3.** Řekneme, že soubor náhodných veličin $\{X_n\}_{n\in\mathbb{N}}$ je *(silně) pseudonáhodný*, pokud je (silně) výpočetně nerozlišitelný od souboru $\{U_{\ell(n)}\}_{n\in\mathbb{N}}$ náhodných veličin s uniformním rozdělením.

Není těžké ukázat, že statisticky blízké soubory jsou (silně) výpočetně nerozlišitelné. Opačně to ale neplatí, jak ukazuje následující věta.

**Věta 15.4.** *Nechť $U_n$ je rovnoměrně rozdělená náhodná veličina s hodnotami $\{0,1\}^n$. Pak existuje soubor náhodných veličin $\{X_n\}_{n\in\mathbb{N}}$ silně výpočetně nerozlišitelný od $\{U_n\}_{n\in\mathbb{N}}$ takový, že*

$$\Delta_{X,U}(n) \xrightarrow{n\to\infty} \frac{1}{2}.$$

*Důkaz.* Položme $m = 2^{\frac{n}{2}}$. Pro každé $n$ najdeme multimnožinu

$$S_n = \{s(i) \mid i = 1, 2, \ldots, m\} \subseteq \{0,1\}^n$$

a definujeme $X_n = X_{S_n} = s(U_m)$. Jinak řečeno, $X_n$ vybírá uniformně náhodně prvek multimnožiny $S_n$. Zřejmě platí

$$\Delta_{X,U}(n) \geq \frac{1}{2} \sum_{v \notin S_n} |\Pr[X_n = v] - \Pr[U_n = v]| = \frac{1}{2} \left| 0 - \left(1 - \frac{2^{\frac{n}{2}}}{2^n}\right) \right| = \frac{1}{2} - \frac{1}{2} \cdot 2^{-\frac{n}{2}},$$

a tedy

$$\Delta_{X,U}(n) \xrightarrow{n \to \infty} \frac{1}{2},$$

protože libovolná statistická diference je nejvýše $\frac{1}{2}$.

Množinu $S_n$ chceme zvolit tak, aby žádný obvod velikosti nejvýše $2^{\frac{n}{8}}$ s $n$ vstupy nerozlišil s nezanedbatelnou pravděpodobností $X_n$ od rovnoměrně rozložené náhodné veličiny. K tomu je nejprve třeba získat nějaký odhad počtu takových obvodů.

*

*Počet booleovských obvodů velikosti nejvýše $k$ s $n$ vstupy.* Odhad provedeme pro $k > n > 10$ (to v našem případě určitě nastává pro $n \geq 64$). Nechť $(k_\wedge, k_\vee, k_\neg)$ určuje počty jednotlivých typů vrcholů. Takových vektorů je méně než $k^3$. Každý vrchol má dva z $k + n$ možných vstupů, což znamená méně než $(k + n)^{2k}$ různých propojení. Celkový počet obvodů je tedy méně než

$$k^3 (k + n)^{2k} < (2k)^{2k+3} = 2^{(2k+3)\log 2k} < 2^{k^2}.$$

*

Mějme nyní nějaký pevně daný obvod $C$. Chceme odhadnout, kolik různých množin $S_n$ je $C$ schopno s nezanedbatelnou pravděpodobností rozpoznat. Přesněji, zkoumejme, pro kolik množin $S_n$ platí

$$|\Pr[C(X_{S_n}) = 1] - \Pr[C(U_n) = 1]| \geq 2^{-\frac{n}{8}}.$$

Zde je $2^{-\frac{n}{8}}$ vhodně zvolená zanedbatelná funkce, shoda s velikostí obvodu je nedůležitá. Zkoumání provedeme pomocí pravděpodobnostního odhadu: zjistíme s jakou pravděpodobností bude obvodem odlišeno $X_{S_n}$, pokud $S_n$ vznikne nezávislými rovnoměrně náhodnými volbami prvků $s_i$. Všimněme si, že pravděpodobnost uvažovaná při volbě $S_n$ je zde pouze prostředkem pro provedení početního argumentu, nesouvisí s náhodností veličiny $X_n$.

Označme $p_C = \Pr[C(U_n) = 1]$ a označme $Z_i = C(s_i) = C(U_n)$, $i = 1, 2, \ldots, 2^{\frac{n}{2}}$. Veličiny $Z_i$ jsou tedy nezávislé náhodné veličiny s Bernoulliho rozdělením se střední hodnotou $p_C$. Vzhledem k tomu, že $X_{S_n}$ vybírá z množiny $S_n$ rovnoměrně náhodně, platí

$$\Pr[C(X_{S_n}) = 1] = \frac{1}{2^{\frac{n}{2}}} \sum_{i=1}^{2^{n/2}} Z_i \,.$$

Máme tedy

$$|\Pr[C(X_{S_n}) = 1] - \Pr[C(U_n) = 1]| = \left| \frac{1}{2^{\frac{n}{2}}} \sum_{i=1}^{2^{n/2}} Z_i - p_C \right|,$$

a z Chebyshevovy nerovnosti dostáváme

$$\Pr\left[ \left| \frac{1}{2^{\frac{n}{2}}} \sum_{i=1}^{2^{n/2}} Z_i - p_C \right| \right] \geq 2^{-\frac{n}{8}} \leq 2 \exp\left(-2 \cdot 2^{\frac{n}{2}} \cdot (2^{-\frac{n}{8}})^2\right) < 2^{-2^{n/4}}.$$

Pro pravděpodobnost $p$, že $X_{S_n}$ bude rozeznáno od alespoň jednoho obvodu velikosti nejvýše $2^{n/8}$, tedy s využitím odhadu na počet takových obvodů platí

$$p < \sum_{|C|<2^{n/8}} 2^{-2^{n/4}} < 2^{(2^{n/8})^2} \cdot 2^{-2^{n/4}} = 1\,.$$

Z toho plyne, že $X_{S_n}$ pro nějaké $S_n$ splňuje

$$|\Pr[C(X_{S_n}) = 1] - \Pr[C(U_n) = 1]| < 2^{-\frac{n}{8}}$$

pro všechny obvody $C$ velikosti nejvýše $2^{\frac{n}{8}}$. Tím je důkaz hotov. □

Definice nerozlišitelnosti počítá s jediným vzorkem příslušné náhodné veličiny. Je proto přirozené se ptát, jak je to s nerozlišitelností nerozlišitelných distribucí, pokud je možné porovnávat různé, nezávislé vzorky. Odpověď není úplně jednoznačná. Je například známo, že existuje soubor náhodných veličin $\{X_n\}$, který je výpočetně nerozlišitelný od $\{U_n\}$ a přitom má nosič $X_n$ (tj. množina hodnot s nenulovou pravděpodobností) velikost pouze $n^2$. To ale znamená, že soubor $\left\{\left(X_n^{(1)}, X_n^{(2)}\right)\right\}$ sestávající z dvojic nezávislých kopií $X_n$ je od $\left\{\left(U_n^{(1)}, U_n^{(2)}\right)\right\}$ rozlišitelný algoritmem, který na vstupu $(z_1, z_2)$ testuje, zda $z_1 = z_2$. Jak ale ukazuje následující věta, náhodnou veličinu $X_n$ s takovou vlastností nelze sestrojit žádným polynomiálním pravděpodobnostním strojem ve smyslu následující definice.

Řekneme, že soubor náhodných veličin $\{X_n\}$ je *polynomiálně konstruovatelný*, pokud existuje polynomiální pravděpodobnostní algoritmus $D$ takový, že náhodné veličiny $X_n$ a $D(1^n)$ mají stejné rozdělení.

**Věta 15.5** (O nerozlišitelnosti opakovanými experimenty)**.** *Nechť jsou $\{X_n\}$ a $\{Y_n\}$ polynomiálně konstruovatelné výpočetně nerozlišitelné soubory náhodných veličin. Pak jsou nerozlišitelné i polynomiálně mnoha opakovanými experimenty. Tj. platí, že pro libovolný polynom $q$ a libovolný polynomiální pravděpodobnostní algoritmus $D$ je*

$$\Delta(n) = \left|\Pr\left[D\left(X_n^{(1)}, \ldots, X_n^{(q(n))}\right) = 1\right] - \Pr\left[D\left(Y_n^{(1)}, \ldots, Y_n^{(q(n))}\right) = 1\right]\right|,$$

*zanedbatelná funkce v $n$, kde $X_n^{(i)}$ jsou vzájemně nezávislé veličiny se stejným rozdělením jako $X_n$ a $Y_n^{(i)}$ jsou vzájemně nezávislé veličiny se stejným rozdělením jako $Y_n$.*

*Důkaz.* Předpokládejme, že rozlišovací výhoda $\Delta(n)$ algoritmu $D$ není zanedbatelná. Pro $i = 1, \ldots, q(n)$ definujme tzv. *hybridní* náhodné veličiny

$$H_n^{(i)} = \left(X_n^{(1)}, \ldots, X_n^{(i)}, Y_n^{(i+1)}, \ldots, Y_n^{(q(n))}\right).$$

Označme $\Delta_i(n)$, $i = 0, 1, \ldots, q(n) - 1$, výhodu, se kterou $D$ rozlišuje dva sousední hybridní soubory, tedy

$$\Delta_i(n) = \left|\Pr\left[D\left(H_n^{(i+1)}\right) = 1\right] - \Pr\left[D\left(H_n^{(i)}\right) = 1\right]\right|.$$

Platí

$$\begin{aligned}
\Delta(n) &= \left|\Pr\left[D\left(H_n^{(q(n))}\right) = 1\right] - \Pr\left[D\left(H_n^{(0)}\right) = 1\right]\right| \\
&= \left|\sum_{k=0}^{q(n)-1} \Pr\left[D\left(H_n^{(k+1)}\right) = 1\right] - \sum_{k=0}^{q(n)-1} \Pr\left[D\left(H_n^{(k)}\right) = 1\right]\right| \\
&\leq \sum_{k=0}^{q(n)-1} \Delta_i(n).
\end{aligned}$$

Z toho plyne, že alespoň jedna výhoda $\Delta_j(n)$ není zanedbatelná.

Předpokládejme na chvíli, že víme, pro které $j$ je $\Delta_j(n)$ nezanedbatelné, a uvažme algoritmus $D'_j$, který na vstupu $z$ spočítá

$$D'_j(z) = D\left(X_n^{(1)}, \ldots, X_n^{(j)}, z, Y_n^{(j+2)} \ldots, Y_n^{(q(n))}\right).$$

Pro výhodu algoritmu $D'$ platí

$$\begin{aligned}\left|\Pr\left[D'_j(X_n) = 1\right] - \Pr\left[D'_j(Y_n) = 1\right]\right| &= \left|\Pr\left[D\left(H_n^{(j+1)}\right) = 1\right] - \Pr\left[D\left(H_n^{(j)}\right) = 1\right]\right| \\ &= \Delta_j(n).\end{aligned}$$

Algoritmus $D'_j$ tedy ukazuje, že $\{X_n\}$ a $\{Y_n\}$ nejsou výpočetně nerozlišitelné.

Protože však $j$ neznáme, zkusíme ho uhodnout. Nechť tedy algoritmus $D''$ nejprve s rovnoměrnou pravděpodobností zvolí $k \in \{0, 1, \ldots, q(n) - 1\}$ a poté pro něj spustí algoritmus $D'_k$. Výhoda algoritmu $D''$ je nejméně rovna výhodě algoritmu $D'_j$ vynásobené pravděpodobností volby $k = j$, tedy celkem

$$\frac{\Delta_j}{q(n)},$$

což je stále ještě nikoli zanedbatelná funkce. Protože jsme ukázali, že z předpokladu rozlišitelnosti polynomiálního počtu vzorků plyne rozlišitelnost jednoho vzorku, je důkaz u konce.

Poznamenejme ještě, že polynomiální konstruovatelnost obou souborů jsme využili při definici algoritmu $D'$, protože ten si musí příslušné kopie $X_n^{(i)}$ a $Y_n^{(i)}$ zkonstruovat sám. □

*Dodatek k důkazu.* Uvedený důkaz motivuje konstrukci algoritmu $D''$, ale počítá jeho výhodu příliš pesimisticky. Ve skutečnosti můžeme uvážit, že platí

$$\Pr\left[D''(X_n) = 1\right] = \frac{1}{q(n)} \sum_{k=1}^{q(n)} \Pr\left[D\left(H_n^{(k)}\right) = 1\right],$$

$$\Pr\left[D''(Y_n) = 1\right] = \frac{1}{q(n)} \sum_{k=0}^{q(n)-1} \Pr\left[D\left(H_n^{(k)}\right) = 1\right],$$

a proto

$$\left|\Pr\left[D''(X_n) = 1\right] - \Pr\left[D''(Y_n) = 1\right]\right| = \frac{\Delta(n)}{q(n)}.$$

□

## 16. Jednosměrné funkce a těžký bit

**Definice 16.1.** Funkce $f : \{0,1\}^* \to \{0,1\}^*$ je *jednosměrná*, pokud je:

- spočitatelná v polynomiálním čase;
- těžko invertovatelná. T.j. pro každý pravděpodobnostní polynomiální algoritmus $B$ je
$$\Pr_{x \sim U_n}\left[B(f(x)) \in f^{-1} \circ f(x)\right]$$
zanedbatelná funkce v $n$.

Zvláštním případem jsou *jednosměrné permutace*, tedy jednosměrné funkce, které zachovávají délku vstupu a jsou prosté.

**Definice 16.2.** Zobrazení $b : \{0,1\}^* \to \{0,1\}$ je *těžký bit* funkce $f$, pokud je:

- spočitatelné v polynomiálním čase;

- není odhadnutelné s nezanedbatelnou výhodou. T.j. pro každý pravděpodobnostní polynomiální algoritmus $B$ je
$$\left| \Pr_{x\sim U_n} [B(f(x)) = b(x)] - \frac{1}{2} \right|$$
zanedbatelná funkce v $n$.

Je-li $f$ prostá, pak má těžký bit, právě když je jednosměrná. Pokud by nebyla jednosměrná, bylo by možné hodnotu $b(x)$ spočítat z $f(x)$ invertováním. Pokud by naopak bylo možné s nezanedbatelnou pravděpodobností spočítat libovolný bit $x_i$, bylo by možné s nezanedbatelnou pravděpodobností najít celé $x$.

Pro funkce, které prosté nejsou je situace složitější. Definice těžkého bitu totiž požaduje, aby bylo nalezeno $b(x)$ pro přesně to $x$, které bylo použito. Pokud má ale např. $f(x)$ dva stejně pravděpodobné vzory $x$ a $x'$ s odlišnými hodnotami $b(x) \neq b(x')$, je odhad nemožný. Příkladem je funkce, která jen maže první bit, který se tak stává jejím těžkým bitem, aniž by funkce byla jednosměrná.

Pro každou jednosměrnou permutaci tedy existuje nějaký bit vzoru, který je těžkým bitem. Dokonce je takových hodně, ale apriori to o žádném z nich nevíme. Stačí uvážit funkci $f'(x, y) = (x, f(y))$, kde $f$ je jednosměrná. Funkce $f'$ je pak také jednosměrná a přitom prozrazuje celou první polovinu vstupu. Poněkud obecněji se lze domnívat, že většina binárních predikátů jednosměrné permutace je těžká. Tato intuice je založena na předpokladu, že z dostatečného množství binárních predikátů řetězce $x$ je možné $x$ zrekonstruovat. Následující věta tuto intuici potvrzuje pro lineární predikáty a libovolnou jednosměrnou funkci, tedy nejen jednosměrnou permutaci.

**Věta 16.3.** *Nechť je $f$ libovolná jednosměrná funkce. Definujme funkci $g$ předpisem*
$$g(x, r) := (f(x), r),$$
*kde $|x| = |r|$. Pak je bodový součin vektorů $x$ a $r$ (modulo 2) těžkým bitem funkce $g$.*

Věta říká, že naprostá většina kontrolních součtů argumentu $x$ je těžkým bitem funkce $f$. Tvrzení lze reformulovat takto: Pokud při náhodné volbě $r$ můžeme z $f(x)$ rychle spočítat hodnotu $\langle x, r\rangle$ s nezanedbatelnou výhodou, můžeme s nezanedbatelnou pravděpodobností zíkat i celé $x$. Vyslovme nejprve toto tvrzení pro fixní $x$.

**Věta 16.4.** *Pro $x$ délky $n$ označme $b_x(r)$ orákulum splňující*
$$\left| \Pr_{r\sim U_n} [b_x(r) = \langle x, r\rangle] - \frac{1}{2} \right| \geq \varepsilon(n).$$
*Existuje pravděpodobnostní algoritmus $A$ pracující v čase* $\operatorname{poly}\left(\frac{n}{\varepsilon(n)}\right)$ *takový, že pro každé $x$ platí*
$$\Pr[A^{b_x}(1^n) = x] \geq \frac{\varepsilon(n)^2}{2n}.$$

*Důkaz, že z Věty 16.4 plyne Věta 16.3.* Podstatou důkazu je ukázat, jednoduchým početním argumentem, že algoritmus odhadující $\langle x, r\rangle$ s nezandebatelnou pravděpodobností, lze použít jako orákulum z Věty 16.4, které má pro nezanedbatelnou část vstupů $x$ nezanedbatelnou výhodu.

Předpokládejme, že $\langle x, r\rangle$ není těžký bit funkce $g$. Označme $G$ algoritmus, který to dokazuje, tedy který vrací $\langle x, r\rangle$ s nezanedbatelnou pravděpodobností. Označme $\gamma$ jeho výhodu, tedy
$$\gamma(n) := \left| \Pr_{x,r\sim U_n} [G(f(x), r) = \langle x, r\rangle] - \frac{1}{2} \right|.$$

Označme si $S_n$ množinu takových $x$ délky $n$, na kterých je výhoda $G$ alespoň $\gamma(n)/2$. Množina $S_n$ má velikost alespoň $\frac{\gamma(n)}{2} \cdot 2^n$, jinak by celková výhoda algoritmu $G$ byla menší než

$$\frac{1}{2^n}\left(\sum_{x\in S_n} 1 + \sum_{x\notin S_n} \frac{\gamma(n)}{2}\right) < \frac{\gamma(n)}{2} + \frac{\gamma(n)}{2} = \gamma(n)\,.$$

Podle Věty 16.4 lze pro každé $x \in S_n$ v čase, který je polynomiální v $n/\gamma(n)$, a s pravděpodobností alespoň $\mathrm{poly}(\gamma(n)/n)$ najít $x$ s pomocí $f(x)$: stačí použít $b_x(r) := G(f(x), r)$. Protože $x \sim U_n$ leží v $S_n$ s pravděpodobností alespoň $\gamma(n)/2$ a protože $\gamma(n)$ není zanedbatelná, dostáváme spor s jednosměrností $f$.

□

*Důkaz Věty 16.4.* Položme $\ell = \lfloor \log_2(n \cdot \varepsilon^{-2}) \rfloor$, kde $\varepsilon = \varepsilon(n)$, a zvolme uniformně náhodně a nezávisle vektory $s_1, s_2, \ldots, s_\ell \in \{0,1\}^n$ (mluvíme o vektorech, protože budeme $\{0,1\}^n$ chápat jako vektorový prostor $\mathbb{F}_2^n$). Dále zvolíme uniformně a nezávisle bity $\sigma_1, \sigma_2, \ldots, \sigma_\ell$. Bit $\sigma_i$ představuje náš tip hodnoty $\langle x, s_i\rangle$. Předpokládejme, že skutečně platí $\sigma_i = \langle x, s_i\rangle$ pro všechna $i = 1, \ldots, \ell$. Pravděpodobnost takového úspěšného tipu je $2^{-\ell} \geq \frac{\varepsilon^2}{n}$. Pro dané $k \in \{1, 2, \ldots, n\}$ se nyní pokusíme spočítat $x_k$, tj. $k$-tý bit vektoru $x$.

Z linearity bodového součinu plyne, že správným odhadem bitů $\sigma_i$ jsme získali hodnoty $\langle x, r\rangle$ pro všechny vektory $r$, které leží v lineárním obalu $s_1, s_2, \ldots, s_\ell$. Je-li $J$ neprázdná podmnožina $\{1, 2, \ldots, \ell\}$, označme $r_J = \bigoplus_{j\in J} s_j$. Bit $\sigma_J = \bigoplus_{j\in J} \sigma_j$ je potom naším odhadem hodnoty $\langle x, r_J\rangle$. Máme tedy polynomiálně mnoho, konkrétně $2^\ell - 1$ (ne nutně různých) vektorů $r_J$ a hodnot $\langle x, r_J\rangle$, které použijeme k nalezení $x_k$ s pomocí rovnosti

$$\langle x, r_J\rangle \oplus \langle x, r_J \oplus e_i\rangle = \langle x, e_k\rangle = x_k,$$

kde vektor $e_i$ je kanonický bázový vektor. Hodnotu $\langle x, r_J\rangle$ podle předpokladu známe, pro zjištění hodnoty $\langle x, r_J \oplus e_k\rangle$ použijeme orákulum $b_x$.

Označme $Z_J$ binární náhodnou veličinu indikující správnost odpovědi orákula $b_x$ při našem dotazu na $r_J$, a tedy i platnost rovnosti

$$\langle x, r_J\rangle \oplus b_x(r_J \oplus e_i) = x_k.$$

Vektor $r_J \oplus e_k$ je díky $s_i \sim U_n$ pro každé $J$ rovnoměrně náhodný, platí tedy $\Pr[Z_J = 1] = 1/2 + \varepsilon$. Pro $K \neq J$ jsou navíc vektory $r_J \oplus e_k$ a $r_K \oplus e_k$ nezávislé, protože se liší o alespoň jedno rovnoměrně a nezávisle zvolené $s_i$.

Máme tedy $m = 2^\ell - 1$ po dvou nezávislých hlasování o hodnotě $x_k$ s převahou správných odpovědí. Zdůrazněme, že $Z_J$ samozřejmě nejsou nezávislá, jsou ale jsou po dvou nezávislá, což stačí k použití Čebyševova odhad pro odhad pravděpodobnosti omylu většinového rozhodnutí:

$$\begin{aligned}\Pr\left[\sum_J Z_J \leq \frac{1}{2}(2^\ell - 1)\right] &\leq \Pr\left[\left|\sum_j Z_j - \left(\frac{1}{2} + \varepsilon\right)\cdot m\right| \geq m\cdot\varepsilon\right] \\ &\leq \frac{m\cdot \mathrm{Var}[Z_J]}{m^2\cdot\varepsilon^2} < \frac{\frac{1}{4} - \varepsilon^2}{\frac{n}{2} - \varepsilon^2} \leq \frac{1}{2n}\,.\end{aligned}$$

Pravděpodobnost, že nedojde k chybě pro žádné $x_k$, $k = 1, 2, \ldots, n$, je tedy alespoň $\left(1 - \frac{1}{2n}\right)^n \geq \frac{1}{2}$. Celková pravděpodobnost nalezení $x$ je tedy alespoň $\frac{1}{2} \cdot \frac{\varepsilon^2}{n}$. □

Důležitým kryptografickým primitivem, pro který se používá těžký bit jednosměrné funkce, je *závazek*. Používá se v situaci, kdy strana $A$ chce předat nějakou zprávu (v definici níže se jedná o jednobitovou zprávu) straně $B$ tak, aby se $B$

nemohla zprávu dozvědět, dokud k tomu nedá $A$ dodatečný souhlas (tajnost), a současně, aby $A$ nemohla zprávu dodatečně změnit (závaznost).

**Definice 16.5.** Řekneme, že polynomiální pravděpodobnostní algoritmus $Z$ je *bitový závazek*, pokud pro každé $m \in \{0,1\}$ a každé $x, x' \in \{0,1\}^*$ platí:

- (závaznost): $Z(x,m) \neq Z(x', 1-m)$
- (tajnost) Soubory náhodných veličin

$$\{Z_{x\sim U_n}(x,0)\}_n \quad \text{a} \quad \{Z_{x\sim U_n}(x,1)\}_n$$

jsou výpočetně nerozlišitelné.

Takovým typem závazku k hodnotě $x$ je pro každou jednosměrnou *prostou* funkci hodnota $f(x)$. Abychom tento fakt využili pro vytvoření bitového závazku stačí použít libovolný těžký bit $b$ funkce $f$. Závazek má potom podobu

$$Z(m,x) = (z_1, z_2) := (f(x), m \oplus b(x)) .$$

Autorizací k otevření závazku je $x$, protože platí $m = z_2 \oplus b(x)$. Ověřením pravosti je rovnost $f(x) = z_1$, která zaručuje i závaznost, protože platí pouze pro jediné $x$.

## 17. Pseudonáhodné generátory

**Definice 17.1.** *(Neuniformně silný) pseudonáhodný generátor* je **deterministický** polynomiální algoritmus $G$, který vstupy délky $n$ prodlužuje na výstupy délky $\ell(n) > n$ tak, že soubor $\{G(U_n)\}_{n\in\mathbb{N}}$ je (silně) pseudonáhodný.

Lze ukázat, že pseudonáhodné generátory existují, právě když existují jednosměrné funkce. Pro jednoduchost budeme uvažovat silnější předpoklad, totiž existenci jednosměrných permutací (obecné tvrzení přináší řadu nepříjemných technických komplikací). Za tohoto předpokladu je snadné sestrojit pseudonáhodný generátor, jak ukazuje následující věta. Zřetězení bitů budeme kvůli lepší čitelnosti značit tečkou.

**Tvrzení 17.2.** *Nechť je $f$ jednosměrná permutace a $b$ její těžký bit. Pak je zobrazení*

$$G : x \mapsto f(x) \cdot b(x)$$

*pseudonáhodný generátor.*

*Důkaz.* Chceme ukázat, že schopnost rozlišit $f(x)\cdot b(x)$ od náhodné posloupnosti délky $|x|+1$ znamená schopnost uhodnout $b(x)$ s pomocí $f(x)$.

Předpokládejme tedy, že $G$ není pseudonáhodný generátor a že $D$ je algoritmus, který to dokazuje, tedy pro který platí

$$\left| \Pr_{x\sim U_n} [D(f(x)\cdot b(x)) = 1] - \Pr_{y'\sim U_{n+1}} [D(y') = 1] \right| = \varepsilon(n),$$

kde $\varepsilon$ není zanedbatelná funkce. Tuto rovnost můžeme pro lepší porozumění přepsat jako

$$\left| \Pr_{y\sim U_n} \left[D\left(y\cdot b\circ f^{-1}(y)\right) = 1\right] - \Pr_{\substack{y\sim U_n \\ \sigma\sim U_1}} [D(y\cdot\sigma) = 1] \right| = \varepsilon(n).$$

Protože s pravděpodobností $1/2$ platí $\sigma = b\circ f^{-1}(y)$, platí také

$$\Pr_{\substack{y\sim U_n \\ \sigma\sim U_1}} [D(y\cdot\sigma) = 1] = \frac{1}{2} \Pr_{y\sim U_n} \left[D\left(y\cdot b\circ f^{-1}(y)\right) = 1\right] + \frac{1}{2} \Pr_{y\sim U_n} \left[D\left(y\cdot \overline{b\circ f^{-1}(y)}\right) = 1\right],$$

odkud dostáváme

$$\left| \Pr_{y\sim U_n} \left[D\left(y\cdot b\circ f^{-1}(y)\right) = 1\right] - \Pr_{y\sim U_n} \left[D\left(y\cdot \overline{b\circ f^{-1}(y)}\right) = 1\right] \right| = 2\,\varepsilon(n) .$$

Rozlišovač $D$ nyní mírně modifikujeme na algoritmus $A$, který bude z $f(x)$ hádat $b(x)$. Na vstupu $y \in \{0,1\}^n$ zvolí algoritmus $A$ uniformně náhodný bit $\sigma$ a poté rozhodne takto:

$$A(y) = \begin{cases} \sigma, & \text{pokud } D(y \cdot \sigma) = 1; \\ \overline{\sigma}, & \text{pokud } D(y \cdot \sigma) = 0. \end{cases}$$

Spočítejme úspěšnost algoritmu $A$.

$$\begin{aligned}
&\Pr_{y \sim U_n}\left[A(y) = b \circ f^{-1}(y)\right] = \\
&= \Pr_{\substack{y \sim U_n \\ \sigma \sim U_1}}\left[D(y \cdot \sigma) = 1 \wedge \sigma = b \circ f^{-1}(y)\right] + \Pr_{\substack{y \sim U_n \\ \sigma \sim U_1}}\left[D(y \cdot \sigma) = 0 \wedge \overline{\sigma} = b \circ f^{-1}(y)\right] = \\
&= \Pr_{\substack{y \sim U_n \\ \sigma \sim U_1}}\left[D(y \cdot b \circ f^{-1}(y)) = 1 \wedge \sigma = b \circ f^{-1}(y)\right] + \\
&+ \Pr_{\substack{y \sim U_n \\ \sigma \sim U_1}}\left[D(y \cdot \overline{b \circ f^{-1}(y)}) = 0 \wedge \sigma = \overline{b \circ f^{-1}(y)}\right] = \\
&= \frac{1}{2} \Pr_{y \sim U_n}\left[D(y \cdot b \circ f^{-1}(y)) = 1\right] + \frac{1}{2} \Pr_{y \sim U_n}\left[D(y \cdot \overline{b \circ f^{-1}(y)}) = 0\right] = \\
&= \frac{1}{2} \Pr_{y \sim U_n}\left[D(y \cdot b \circ f^{-1}(y)) = 1\right] + \frac{1}{2}\left(1 - \Pr_{y \sim U_n}\left[D(y \cdot \overline{b \circ f^{-1}(y)}) = 1\right]\right) = \\
&= \frac{1}{2} + \frac{1}{2}\left(\Pr_{y \sim U_n}\left[D\left(y \cdot b \circ f^{-1}(y)\right) = 1\right] - \Pr_{y \sim U_n}\left[D\left(y \cdot \overline{b \circ f^{-1}(y)}\right) = 1\right]\right) = \\
&= \frac{1}{2} \pm \varepsilon(n)\,.
\end{aligned}$$

Výhoda

$$\left|\Pr_{y \sim U_n}\left[A(y) = b \circ f^{-1}(y)\right] - \frac{1}{2}\right|$$

algortimu $A$ tedy není zanedbatelná funkce, což je spor s předpokladem, že $b$ je těžký bit funkce $f$. □

Shrňme ještě jednou sdělení předchozí věty. Těžký bit se chová jako nový náhodný bit, přestože je dán z prvních $n$ bitů deterministicky. Jako náhodný se chová právě proto, že je těžký: jeho (jednoznačná) souvislost s předchozím vstupem totiž není algoritmicky detekovatelná s nezanedbatelnou výhodou.

Nevýhodou právě zkonstruovaného pseudonáhodného generátoru je jeho krátkost, produkuje pouze jediný nový náhodný bit. Stačí ale myšlenku konstrukce libovolně (polynomiálně-krát) vhodným způsobem opakovat, abychom dostali libovolný (polynomiální) počet náhodných bitů. Jako v případě nerozlišitelnosti nerozlišitelných distribucí opakovanými experimenty, ani v tomto případě není opakováním narušena pseudonáhodnost. Důkaz používá techniku hybridních distribucí podrobně vyloženou v důkazu nerozlišitelnosti opakovanými experimenty.

**Tvrzení 17.3.** *Nechť je $G$ pseudonáhodný generátor s prodlužovací funkcí $\ell(n) = n + 1$ a nechť $\ell'$ je libovolný polynom. Definujme zobrazení $G'$ předpisem*

$$G'(s) = \sigma_1(s) \cdot \sigma_2(s) \cdots \sigma_{\ell'(|x|)}(s),$$

*kde*

$$\begin{aligned}
x_0 &= s, \\
G(x_{i-1}) &= x_i \cdot \sigma_i(s), \quad i = 1, 2, \ldots, \ell'(|x|)\,.
\end{aligned}$$

*Pak je $G'$ pseudonáhodný generátor.*

*Důkaz.* Nechť $X_1, X_2, \ldots, X_{\ell(n)}$ označují nezávislé kopie náhodné proměnné $U_1$. Chceme ukázat, že nelze s nezanedbatelnou výhodou rozlišit $\big(X_1, \ldots, X_{\ell(n)}\big)$ od $\big(\sigma_1(s), \ldots, \sigma_{\ell(n)}(s)\big)$ pro $s \sim U_n$. Zdůrazněme, že $\sigma_i(s)$ jsou náhodné veličiny závislé na uniformní volbě téhož $s$, a jsou tedy zcela závislé.

Definujme hybridní distribuce

$$H^{(i)} = \big(X_1, X_2, \ldots, X_i, \sigma_1(s), \sigma_2(s), \ldots, \sigma_{\ell(n)-i}(s)\big)\,.$$

Algoritmus $D$, který rozlišuje $H^{(0)}$ od $H^{\left(\ell'(n)\right)}$ s nezandebatelnou výhodou, rozlišuje s nezanedbatelnou výhodou také nějaké $H^{(i)}$ od $H^{(i+1)}$. Ukažme, že to umožňuje s nezanedbatelnou výhodou rozlišit $G(x)$ od $U_{n+1}$ ve sporu s předpokladem, že $G$ je pseudonáhodný generátor.

Předpokládejme, že jsme zvolili správné $i$, což se stane s nezanedbatelnou pravděpodobností $1/\ell'(n)$. Pro dané $y \cdot \sigma$, kde $|y| = n$ a $|\sigma| = 1$, nyní necháme algoritmus $D$ rozhodnout vstup

$$\big(X_1, X_2, \ldots, X_i, \sigma, \sigma_1(y), \sigma_2(y), \ldots, \sigma_{\ell(n)-i-1}(y)\big)\,.$$

Je-li $y \cdot \sigma \sim U_{n+1}$ jedná se o $H^{(i+1)}$. Je-li naopak $y \cdot \sigma = G(s)$ pro $s \sim U_n$, jedná se o $H^{(i)}$, protože $\sigma = \sigma_1(s)$ a $\sigma_i(y) = \sigma_{i+1}(s)$. Tím je důkaz hotov. □

## 18. Průměrná a generická složitost

Chceme-li definovat průměrnou složitost algoritmu $A$, potřebujeme na jeho definičním oboru $D_A$ zavést pravděpodobnostní míru. Jednou z možností je funkce

$$\mu(x) = \frac{1}{d_{|x|} \cdot |x| \cdot (|x|+1)},$$

kde

$$d_n = |D_A \cap \{0,1\}^n|$$

označuje počet slov délky $n$ ležících v definičním oboru. Tato míra splňuje

$$\sum_{x \in D_A} \mu(x) = 1$$

a také $\mu(x) = \mu(y)$, pokud $|x| = |y|$.

Pro daný algoritmus $A$ označme $t_A(x)$ délku výpočtu na vstupu $x$.

**Definice 18.1.** Řekneme, že funkce $t_A(x)$ je v průměru polynomiální, pokud existuje $\varepsilon > 0$ takové, že

$$\sum_{x \in D} \mu(x) \frac{t_A(x)^\varepsilon}{|x|} < \infty.$$

Řekneme, že problém má v průměru polynomiální složitost, pokud pro jeho řešení existuje algoritmus, jehož časová náročnost je v průměru polynomiální.

Funkce

$$f(x) = \begin{cases} 2^{|x|}, & \text{pokud } x \in 1^*, \\ n^2 & \text{jinak.} \end{cases}$$

Tato funkce je exponenciální v nejhorším případě, ale snadno ověříme, že je polynomiální v průměru. Naproti tomu funkce

$$f'(x) = \begin{cases} 2^{2^{|x|}}, & \text{pokud } x \in 1^*, \\ n^2 & \text{jinak.} \end{cases}$$

je exponenciální i v průměru.

Průměrná složitost bere ohled na výjimečné obtížné případy s ohledem na míru jejich četnosti. Přesto vidíme, že i jediný obtížný případ v dané délce, pokud je skutečně velmi obtížný, může být na překážku polynomiální složitosti problému.

Pokud předpokládáme, že instance jsou generovány náhodně a uniformně, bylo by rozumnější na výjimečné případy nebrat ohled vůbec, pokud pravděpodobnost jejich výskytu je zanedbatelná. Při invertování jednosměrné funkce je např. zanedbatelná pravděpodobnost neúspěchu přípustná a obtížnost jednoho případu pro každou délku je tedy irelevantní, bez ohledu na míru jeho obtížnosti (výpočet může být i nekonečný). Tyto úvahy vedou k definici generické množiny.

Uvažujme nějakou podmnožinu $D$ množiny $S$. Označme

$$D_n = D \cap \{0,1\}^n \qquad\qquad S_n = S \cap \{0,1\}^n$$

množiny slov dané délky. Relativní hustota množiny $D$ v $S$ pro dané $n$ je

$$\rho_n = \frac{|D_n|}{|S_n|}.$$

Řekneme, že množina $D$ je *generická* v $S$ pokud

$$\lim_{n\to\infty} \rho_n = 1.$$

Řekneme, že je *silně generická* pokud posloupnost konverguje exponenciálně rychle, tj. pokud existuje $0 < \sigma < 1$ takové, že pro dostatečně velká $n$ platí

$$1 - \rho_n < \sigma^n.$$

Nyní můžeme říct, že problém má na $S$ *(silně) generickou složitost* odpovídající složitostní třídě $\mathcal{C}$, pokud má složitost $\mathcal{C}$ na nějaké (silně) generické podmnožině $D \subset S$. Přesněji to znamená, že existuje algoritmus $A$ definovaný na $S$, který

a) řeší všechny instance $x \in D$ s náročností odpovídající $\mathcal{C}$,
b) pro žádnou instanci $x \in S \setminus D$ neposkytne chybné řešení.

## 19. Důkazové systémy

**Definice 19.1.** *Interaktivní Turnigův stroj* je vícepáskový **pravděpodobnostní** Turingův stroj se vstupem a výstupem, který kromě vstupní pásky (kterou nazýváme *veřejná*), výstupní pásky a pracovních pásek obsahuje ještě

- dodatečnou vstupní pásku, kterou nazýváme *soukromá*,
- *vstupní komunikační* pásku, která je jen ke čtení,
- *výstupní komunikační* pásku, která je jen ke psaní,
- *stavový bit*, tj. „pásku“ sestávající z jednoho políčka, které nese nulu nebo jedničku.

Stroji je přiřazena *identita* nula nebo jedna.

Program obsahuje instrukce pouze pro případ, že stavový bit je roven identitě stroje. V takovém případě říkáme, že stroj pracuje, v opačném případě je *nečinný*.

Množinu interaktivních Turingových strojů označme **ITM** Definice interaktivního stroje, stejně jako jeho název, ukazuje, že smysl jeho výpočtu spočívá v komunikaci s jiným interaktivním strojem.

**Definice 19.2.** *Interaktivní systém* je dvojice interaktivních Turingových strojů $(A, B)$, které

- mají opačnou identitu,
- sdílejí veřejnou vstupní pásku,
- sdílejí stavový bit,
- vstupní komunikační páska stroje $A$ je výstupní komunikační páskou stroje $B$ a naopak.

Díky opačné identitě strojů je vždy jeden ze strojů interaktivního systému nečinný a jeden pracuje. Výpočet interaktivního systému tedy sestává z několika *fází*, což jsou souvislé úseky, ve kterých se stavový bit nemění.

Ve chvíli, kdy pracující stroj přepíše stavový bit, dostane se do nečinnosti a začíná další fáze, ve které pracuje druhý stroj až do chvíle, kdy přepíše stavový bit zpět. Stroje si takto vzájemně „udílejí slovo“.

Pro posuzování činnosti interaktivního systému jsou důležité následující konvence:

- Výstupem interaktivního výpočtu je výstup stroje $B$.
- Náročnost výpočtu je posuzována pouze nároky stroje $B$. Časová náročnost je tedy počet kroků, které provedl stroj $B$ (když pracoval).

Asymetrická definice náročnosti interaktivního systému souvisí s tím, že tyto systémy chápeme jako komunikaci mezi *dokazovatelem* (stroj A) a *ověřovatelem* (stroj B) tvrzení o přítomnosti vstupu $x$ v rozhodovaném jazyku $L$. Proto je také nazýváme *interaktivní důkazové systémy.*

Na dokazovatele neklademe žádná omezení, pokud jde o výpočetní sílu, můžeme tedy předpokládat, že $A$ je schopen ověřit, zda $x \in L$ (za minimálního předpokladu, že $L$ je rekursivní). Pokud by tedy ověřovatel mohl spoléhat na informaci poskytovatele, bylo by snadné rozhodovat libovolný rekursivní jazyk.

Smysl interaktivních důkazových systému však spočívá v tom, že ověřovatel je informaci týkající se vstupu schopen ověřit. Vyžaduje tedy od dokazovatele důkaz pro jeho tvrzení. Od systému vyžadujeme tři vlastnosti:

- *efektivita* (*efficiency*): ověřovatel pracuje rychle;
- *úplnost* (*completeness*): ověřovatel přijme platný důkaz;
- *spolehlivost* (*soundness*): ověřovatel nepřijme neplatný důkaz od žádného dokazovatele, který se ho snaží přesvědčit o nesprávném tvrzení.

Tento neformální popis vede k následující definici složitostní třídy **IP**.

**Definice 19.3.** Jazyk $L$ leží v **IP**, pokud existuje interaktivní systém $(A, B)$, který

- (efektivita) pracuje v polynomiálním čase;
- (úplnost) systém $(A, B)$ přijme vstup $x \in L$ s pravděpodobností alespoň $\frac{2}{3}$;
- (spolehlivost) pokud $x \notin L$, pak pro libovolný interaktivní stroj $A^*$ je pravděpodobnost, že systém $(A^*, B)$ přijme $x$, menší než $\frac{1}{3}$.

**Tvrzení 19.4.**

(1) $\mathbf{BPP} \subseteq \mathbf{IP}$
(2) $\mathbf{NP} \subseteq \mathbf{IP}$

*Důkaz.*

(1) Nechť $L \in \mathbf{BPP}$ a stroj $B$ ho rozhoduje ve smyslu definice **BPP**. Pak interaktivní systém $(\emptyset, B)$ rozhoduje $L$ ve smyslu Definice 19.3, kde $\emptyset$ značí, že výpočet proběhne v jediné fázi. $B$ je schopen rozhodnout $L$ bez pomoci dokazovatele.
(2) Nechť $L \in \mathbf{NP}$. Interaktivní systém, který rozhoduje $L$ pracuje na vstupu $x$ následovně. Výpočet zahajuje stroj $A$, který sdělí stroji $B$ slovo $y$, které je svědkem pro $x$. Stroj $A$ poté ověří, že $(x, y) \in R_L$, kde $R_L$ je svědecká relace $L$.

    Postup je efektivní, neboť $R_L$ je v **P**. Úplnost je dána existencí svědka a skutečností, že $A$ ho může díky své neomezené výpočetní síle nalézt. Spolehlivost plyne z toho, že pro $x \notin L$ žádný svědek neexistuje a tudíž $B$ nepřijme, ať je zpráva od $A$ jakákoli.

□

Všimněte si, že oba stroje během výpočtu rozhodujícího $L \in \mathbf{NP}$ pracují deterministicky.

Bez důkazu uvádíme větu, která třídu **IP** uvádí do jasné souvislosti s deterministickými třídami.

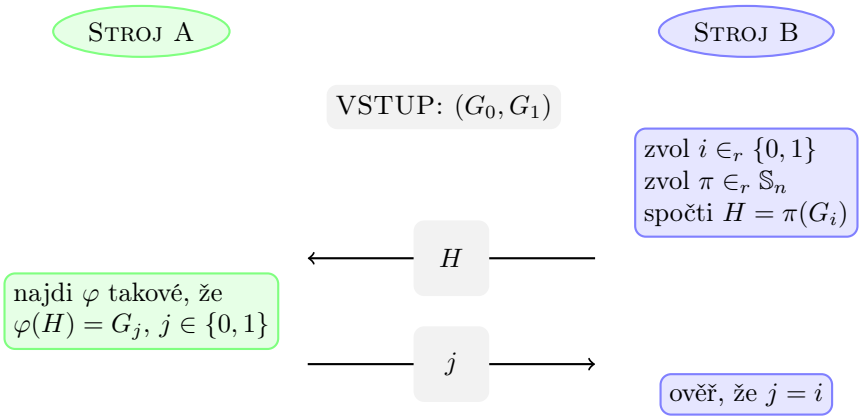
**Věta 19.5** (Shamir)**. IP = PSPACE**

Důkazy pro **BPP** a **NP** jsou snadné. Ukážeme důkaz pro *grafový neisomorfismus*, což je problém, o kterém není známo, zda leží v **NP** nebo v **BPP**.

Budeme předpokládat, že vrcholy grafu jsou čísla $\{1, \ldots, n\}$. Grafy $G_1$ a $G_2$ jsou isomorfní, pokud mají stejný počet prvků $n$ a existuje $\pi \in \mathbb{S}_n$ taková, že $(i, j)$ je hrana $G_1$, právě když $(\pi(i), \pi(j))$ je hrana $G_2$. Potom píšeme $G_1 \cong G_2$ a $\pi(G_1) = G_2$. Grafový neisomorfismus je tedy jazyk

$$\textsc{GraphNI} = \{(G_1, G_2) \mid G_1 \ncong G_2\}.$$

Jádro interaktivního důkazu je popsáno následujícím schématem. Zápisem $x \in_r X$ budeme značit skutečnost, že $x$ je uniformně náhodně zvolený prvek množiny $X$.




**Tvrzení 19.6.** $\textsc{GraphNI} \in \mathbf{IP}$

*Důkaz.* Uvažujme dvě kola výše uvedené komunikace, přičemž stroj $B$ (ověřovatel) přijme, pokud v obou kolech platí $i = j$. Ukážeme, že přijímá GraphNI. Efektivita je zřejmá.

Nechť $(G_1, G_2) \in \textsc{GraphNI}$. Graf $H$ je tedy isomorfní s $G_i$ a nikoli s $G_{1-i}$. Pokud tedy $\varphi(H) = G_j$, platí $i = j$. Stroj $B$ tedy vstup přijme (s pravděpodobností 1). Tím je dokázána úplnost systému.

Předpokládejme nyní $(G_1, G_2) \notin \textsc{GraphNI}$, tedy $G_1 \cong G_2$. Nechť $M_i$, $i \in \{0, 1\}$, je multiset

$$M_i = \{\pi(G_i) \mid \pi \in \mathbb{S}_n\}.$$

(Multiset je množina, která může stejný prvek obsahovat vícekrát, tedy $|M_i| = n!$, přestože některé permutace mohou definovat tentýž graf.) Snadno nahlédneme, že multisety $M_1$ a $M_2$ jsou identické a $H$ je náhodně uniformně zvolený prvek $M = M_1 = M_2$. Pro jakýkoli stroj $P^*$ bude tedy náhodná veličina $j$ nezávislá na volbě $i$. Vzhledem k tomu, že $i$ je voleno náhodně uniformně, je pravděpodobnost $i = j$ přesně $\frac{1}{2}$. Pravděpodobnost přijetí vstupu je tedy $\frac{1}{4}$. Tím je dokázána spolehlivost systému. □

Zajímavou vlastností předchozího důkazu je fakt, že stroj $B$, přestože výpočtem s velkou pravděpodobností (kterou lze opakováním exponenciálně přiblížit jedné) ověří, že $x \in L$, nezíská tím žádnou informaci, kterou by bylo možné $x \in L$ podpořit,

použitelnou v situaci, kdy se $B$ octne v roli dokazovatele. Ověřovatel se nedozví nic, co by sám nemohl spočítat, kromě samotného faktu $x \in L$.

Interaktivní výpočet mající tuto pozoruhodnou vlastnost se nazývá *důkaz s nulovou znalostí*. Formálně je zachycen následující definicí.

**Definice 19.7.** Interaktivní důkazový systém $(A, B)$ se nazývá *důkaz s nulovou znalostí* (zero knowledge proof) pro jazyk $L$, pokud rozhoduje jazyk $L$ ve smyslu Definice 19.3 a navíc pro každý interaktivní stroj $B^*$ existuje (obyčejný) polynomiální pravděpodobnostní stroj $M \in \mathbf{PTM}$ takový, že pro každé $x \in L$ platí:

- S pravděpodobností nejvýše $\frac{1}{2}$ bude výstupem stroje $M$ speciální symbol $\perp$. Řekneme, že výpočet stroje $M$ *selhal*.
- Pokud $M(x) \neq \perp$, pak

$$M(x) \sim \sigma_F(A, B^*)(x),$$

  kde $\sigma_F(A, B^*)(x)$ je závěrečný snímek stroje $B^*$ po výpočtu interaktivního důkazového systému $(A, B^*)$ na vstupu $x$.

Předchozí definice je jakýmsi polotovarem, který bude dokončen upřesněním významu symbolu $\sim$. Uvědomme si, že $\{M(x)\}_{x\in L}$ a $\{\sigma_F(A, B^*)(x)\}_{x\in L}$ jsou soubory náhodných veličin daných náhodným chováním strojů $M$, $A$ a $B^*$.

Symbolu $\sim$ může odpovídat jeden ze stupňů blízkosti distribucí.

- Znamená-li $\sim$ rovnost souborů náhodných veličin, dostáváme definici *důkazu s dokonale nulovou znalostí*.
- Znamená-li $\sim$ statistickou blízkost souborů náhodných veličin, dostáváme definici *důkazu s téměř dokonale nulovou znalostí*.
- Znamená-li $\sim$ výpočetní nerozlišitelnost souborů náhodných veličin, dostáváme definici *důkazu s výpočetně nulovou znalostí*.

Poznamenejme, že na rozdíl od Kapitoly 15 jsou soubory indexovány slovy. Argumentem zanedbatelné funkce je v tomto případě jako obvykle $|x|$.

Třídu jazyků, pro něž existuje důkaz s dokonale nulovou znalostí, značíme **PZK**, třídu jazyků, pro něž existuje důkaz s téměř dokonale nulovou znalostí značíme **SZK** a třídu jazyků, pro něž existuje důkaz s výpočetně nulovou znalostí značíme **CZK**.

Stroj $M$ v definici důkazu s nulovou znalostí se nazývá *simulátor*. Je to stroj, který na vstupu $x \in L$ je schopen pro libovolného ověřovatele $B^*$ – tedy i takového, který se ve snaze o získání nějaké informace nechová podle pravidel systému $(A, B)$ – vypsat jeho závěrečný snímek. Ten mimo jiné obsahuje komunikační pásky, a tedy veškerou informaci, kterou získal $B^*$ od $A$. Tím je uchopena myšlenka nulové znalosti. Cokoli, co se $B^*$ od $A$ dozvěděl si mohl vygenerovat sám pomocí simulátoru $M$.

Nabízí se otázka, zda tedy $M$ neumí sám rozhodovat jazyk $L$. Jako odpověď je třeba zdůraznit, že simulátor je úspěšný **pouze na vstupech** $x \in L$, tedy za předpokladu oné jediné informace, kterou mu má důkaz poskytnout. Pokud $x \notin L$, může se $M$ chovat zcela libovolně.

Nyní můžeme vyslovit tvrzení, které bylo motivací naší definice důkazu s nulovou znalostí.

**Tvrzení 19.8.**

$$\textsc{GraphNI} \in \mathbf{PZK}.$$

*Důkaz.* Popíšeme simulátor $M$ ověřovatele $B^*$. Stroj $M$ se může chovat jako $B^*$ s tou výjimkou, že si musí sám generovat zprávy od $A$. Za předpokladu $(G_0, G_1) \in \textsc{GraphNI}$ je však simulace zpráv $A$ snadná: stroj $A$ vždy pošle $j$, které je rovno $i$, což je hodnota stroji $M$ známá z předchozí simulace. □

Příklad GraphNI je zvláštní tím, že $A$ svou znalost čerpá ze své neomezené výpočetní síly. Vlastně tedy neexistuje žádný důkaz, který by mohl být prozrazen. Následující tvrzení ukazuje, že existují i jazyky, které jsou v **NP** a současně v **PZK**.

V tomto případě si můžeme představovat, že dokazovatel má pro daný vstup $x$ k dispozici na svém soukromém vstupu svědka $y$. Je tedy schopen ověřit $x \in L$ v polynomiálním čase. Jako důkaz ovšem $y$ použít nemůže, protože by se zjevně nejednalo o důkaz s nulovou znalostí.

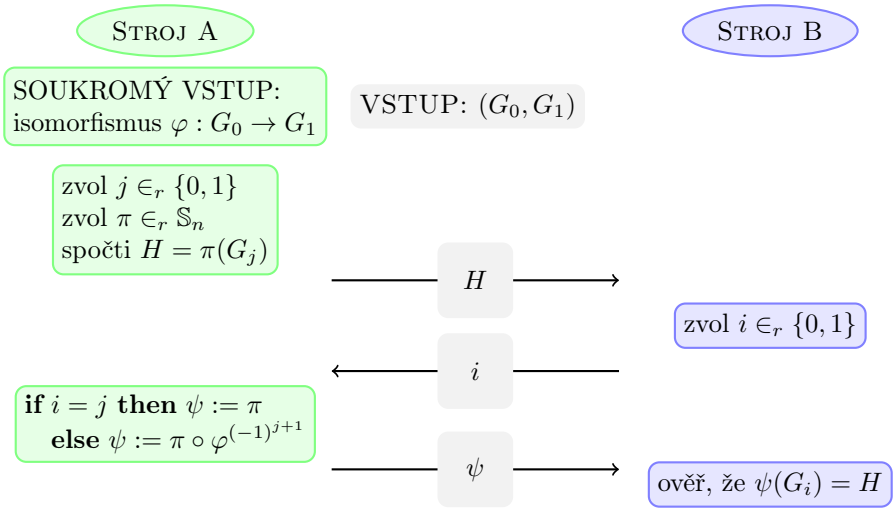
Příkladem bude komplement GraphNI, tedy problém grafového isomorfismu

$$\text{GraphISO} = \{(G_0, G_1) \mid G_0 \cong G_1\}.$$

**Tvrzení 19.9.**

$$\text{GraphISO} \in \mathbf{PZK}$$

*Důkaz.* Popíšeme důkaz s dokonale nulovou znalostí pro GraphISO. Důkaz spočívá ve dvojím opakování následující komunikace:




Stroj $B$ přijme, pokud v obou kolech $\psi(G_i) = H$.

Systém je zřejmě efektivní a úplný (platný vstup přijímá s pravděpodobností 1).

Předpokládejme, že $G_0$ a $G_1$ nejsou isomorfní. Pak je zpráva $H$, bez ohledu na postup $A^*$, isomorfní nejvýše jednomu z grafů. S pravděpodobností alespoň $\frac{1}{2}$ tedy ověřovatel $B$ zvolí $i$, pro které neexistuje žádné $\psi$ splňující požadavek $\psi(G_i) = H$. V takovém případě $B$ odmítne. Ve dvou kolech tedy odmítne s pravděpodobností alespoň $\frac{3}{4}$. Tím je dokázána spolehlivost.

Zbývá ukázat nulovou znalost. Nechť $G_0 \cong G_1$. Simulátor $M$ může simulovat celý výpočet s výjimkou situace, kdy $A$ volí $\psi = \pi \circ \varphi^{-1}$, tedy v situaci, kdy $i \neq j$. Vzhledem k tomu, že $A$ postupuje podle pravidel, má $B^*$ při volbě $i$ k dispozici náhodný prvek multisetu

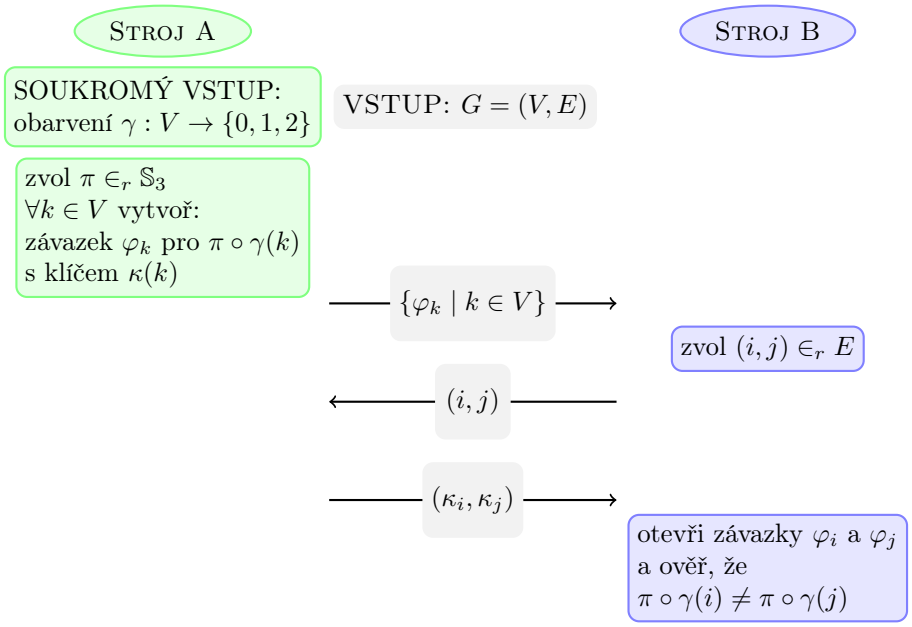
$$M = \{\pi(G_i) \mid \pi \in \mathbb{S}_n\}$$

nezávisle na náhodné hodnotě $j$ (podobně jako v důkazu Tvrzení 19.8). Pravděpodobnost, že zvolí $i \neq j$, je tedy přesně $\frac{1}{2}$. S pravděpodobností $\frac{1}{2}$ tedy $M$ není schopen dokončit simulaci jednoho kola a jeho výpočet selže. Celková pravděpodobnost simulace dvoukolového algoritmu je tedy $\frac{1}{4}$. Stačí tři opakování pokusu o simulaci, aby se pravděpodobnost úspěchu dostala nad $\frac{1}{2}$. $M$ tedy opakuje celý postup třikrát a vytiskne $\perp$ pouze v případě, že všechny pokusy selhaly. Jinak vytiskne výstupní snímek $B^*$ po úspěšném pokusu. □

Pokud existují jednosměrné funkce (permutace), důkazy s nulovou znalostí existují pro všechny jazyky v **NP**. Musíme ovšem slevit z dokonale nulové na výpočetně nulovou znalost.

**Věta 19.10.** *Pokud existují jednosměrné permutace, pak* **NP** $\subset$ **CZK**.

*Nástin důkazu.* Důkaz této věty neuvedeme v úplnosti. Ukážeme však důkaz s nulovou znalostí pro **NP**-úplný jazyk 3Coloring. Jedno kolo interaktivního výpočtu vypadá takto:




Procedura se opakuje $2|E|$ krát. Stroj přijme pokud podmínka je splněna ve všech kolech, jinak odmítne.

Efektivita i úplnost (s pravděpodobností jedna) se ověří snadno.

Ověřme spolehlivost. Nechť graf nemá obarvení třemi barvami. Pak je závazek $(\varphi_1, \ldots, \varphi_{|V|})$ nesprávný (nesmyslný nebo nesplňující podmínku obarvení) pro alespoň jednu hranu. Pravděpodobnost, že to stroj $B$ odhalí je tedy alespoň $\frac{1}{|E|}$. Po $2|E|$ opakováních je pravděpodobnost nesprávného přijetí

$$\left(1 - \frac{1}{|E|}\right)^{2|E|},$$

což je menší než $\frac{1}{4}$.

Naznačme důkaz nulové znalosti. Simulátor $M$ zvolí náhodně uniformně hranu $(i', j') \in E$ a dvojici $(c_1, c_2)$ dvou rozdílných barev. Vytvoří závazek $\varphi_{i'}$ a $\varphi_{j'}$ pro hodnoty $c_1$ a $c_2$. Závazky $\varphi_k$, $k \neq i', j'$ zvolí libovolně. Tím simuluje činnost stroje $A$.

Poté simuluje činnost stroje $B^*$. Ta může být jakákoli, ale končí výstupem $(i, j) \in E$, který $B^*$ posílá stroji $A$. Pokud $(i, j) \neq (i', j')$, simulace selže. V opačném případě vrátí $M$ příslušné klíče a dokončí simulaci jednoho kola.

Pravděpodobnost úspěšnosti simulace jednoho kola je $\frac{1}{|E|}$. Polynomiálně mnoho opakování tedy díky Chernoffovu odhadu zaručuje více než poloviční pravděpodobnost alespoň $|E|$ úspěšných simulací.

Zbývá ověřit, že výstup simulátoru je výpočetně nerozlišitelný od závěrečného snímku skutečného výpočtu. Je zřejmé, že důkaz není s dokonale nulovou znalostí,

není ani s téměř dokonale nulovou znalostí. Závazek $\varphi'$ stroje $M$ se totiž od závazku $\varphi$ stroje $A$ statisticky velmi liší. Závazek $\varphi'$ je závazkem z velké části náhodných hodnot, zatímco závazek $\varphi$ je závazkem malé podmnožiny hodnot - korektních obarvení grafu $G$.

Náhodné veličiny $\varphi$ a $\varphi'$ jsou ovšem výpočetně nerozlišitelné díky tomu, že závazky různých hodnot jsou od sebe výpočetně nerozlišitelné z definice. Důkaz věty by vyžadoval komplikovaný detailní důkaz tohoto tvrzení.

Dále je nutno dokázat, že pokud existuje důkaz s nulovou znalostí pro nějaký **NP**-úplný jazyk, existuje pro všechny jazyky v **NP**, že tedy redukce zachovává vlastnost nulové znalosti. □