Složitost a NP-úplnost

Verze: 2. června 1998

Tento učební text začal vznikat v létě roku 1994, kdy jsem poprvé přednášel „Složitost a NP-úplnost“. Přednáška byla inspirována především knihou [1] Structural Complexity I. Značnou část látky jsem také samozřejmě převzal po Mirko Křivánkovi, který tento předmět přednášel v předchozích letech.

V roce 1995 jsem připsal kapitolu „Na hranici vyčíslitelnosti“, pokrývající některé státnicové otázky nepokryté knihou Structural Complexity I. Tato kapitola byla inspirována 12 kapitolou knihy [2] Introduction to Automata Theory and Computing. Později bylo z této kapitoly přesunuto translační lemma do kapitoly „Věty o hierarchii“. Kromě toho jsem připsal kapitolu „Jiná interpretace ...“, která je mým vlastním zobecněním pojmu nedeterministického Turingova stroje omezeného časem. Toto zobecnění umožnilo popsat pravděpodobnostní třídy a později například hry Artuše s Merlinem stejným jazykem.

V roce 1996 jsem se zabýval především sepisováním disertace, která měla s uvedenou přednáškou málo společného. Přesto když se ke mně donesla informace o vylepšení prostorové věty o hierarchii i pro nedeterministické stroje, neváhal jsem tuto větu připsat. Obdobně jsem postupoval s větou o nedeterministické časové simulaci a hodlám tak postupovat i nadále.

V roce 1997 jsem dále rozvinul pojem „typ“ Turingova stroje a konečně jsem připsal i kapitoly týkající se Booleovských obvodů a P-log-úplnosti.

Nyní v roce 1998 jsem poprvé odpřednášel důkaz Blumovy věty o zrychlení. Zároveň mne to donutilo dopsat důkaz až do konce. To s sebou neslo potřebu stanovit odhady multiplikativních konstant jak ve větách o simulacích, tak ve větě o počtu konfigurací.

Blumova věta o zrychlení byl poslední velký obsahový nedostatek, který jsem si uvědomoval. K dopsání skript nyní již zbývá jediné — definovat základní pojmy, tedy to, co jsem dosud nechával raději na ústní projev. Důvodem je, že jsem se pokoušel uvedené pojmy „nezakonzervovat“. Tyto pojmy se totiž s časem vyvíjí, každý autor je má definovány nepatrně jinak. Z výkladu by mělo být poznat, co je základním neměnným principem. Kvůli možným odlišnostem jsem raději na přednáškách a cvičeních předkládal několik interpretací a ukazoval jsem, zda je změna interpretace podstatná. Základní pojmy jsou také v pozdějších přednáškách drobně doplňovány (orákula, jiná interpretace ...). Nemyslím si, že je tato schizofrenie na škodu. Myslím si, že dobrý programátor by měl ovládat principy programování, ale neměl by být úzce vázán na jeden konkrétní procesor/programovací jazyk/výpočetní model.

V Jílovém u Prahy dne 16.dubna 1998 Vladan Majerech

Cílem přednášky „Složitost a NP-úplnost“ je popsat hranice toho, co je na počítačích možno počítat. Na rozdíl od přednášky „Vyčíslitelnost“ nás zajímá, co je možno počítat při daném časovém nebo prostorovém omezení.

V první polovině přednášky se zabýváme vztahy mezi třídami jazyků definovaných na základě takovýchto omezení. V druhé polovině se zabýváme možnostmi rozšíření výpočetního modelu, obecnějšími třídami jazyků a nejtěžšími jazyky v takových třídách. Pojem nejtěžšího jazyka nám několikrát pomůže dokázat totožnost různým způsobem definovaných tříd.

Koncenzem je, že rozumným časovým omezením je libovolný polynom ve velikosti zadání. V přednášce se zabýváme tím, kam až můžeme při takovém omezení rozšířit naše výpočetní možnosti, stačí-li nám výsledek s malou pravděpodobností chyby.

Jiným směrem rozšiřování výpočetního modelu je „paralelní programování“. Za rozumné časové omezení při paralelním programování je považován polynom logaritmu velikosti vstupu. Tento text se paralelismu pouze okrajově dotýká kapitolami o Booleovských obvodech.

Pokud jste někdy pracovali s rozsáhlými databázemi, pak se Vám nejspíš časové omezní polynomiální ve velikosti dat zdá příliš velké. Takovéto problémy jsou ale zcela mimo rámec této přednášky. Důležité při jejich řešení je udržování vhodné „dynamické datové struktury“. Rozumným časovým omezením pro odpověď dynamické datové struktury je polynom logaritmu velikosti reprezentovaných dat.

Obsah

Seznam obrázků

Seznam Algoritmů

Literatura:

[1] J. L. Balcázar, J. Díaz, J. Gabarró: "Structural Complexity I", *Springer-Verlag Berlin Heidelberg New York London Paris Tokyo 1988*

[2] Hopcroft, Ullmann "Introduction to Automata Theory and Computing", ISBN 0-201-02988-X

1 Připomenutí pojmů (29. září 1997)

1.1 Turingův stroj

V této kapitole by měl být definován Turingův stroj, čtenáře odkazuji na literaturu (José Luis Balcázar, Josep Díaz, Joaquim Gabarró — Structural Complexity I).

Turing Machine

Upozorňuji též na jiné výpočetní modely (RAM, Pointer machine).

Měl by být definován vícepáskový (případně nedeterministický) Turingův stroj. Není přesně specifikován výsledek výpočtu, ale měl by být podrobně vysvětlen průběh výpočtu.

Konfigurace Turingova stroje — Na každé pásce souvislý úsek symbolů páskové abecedy, jinde blank, na každé pásce buď jedna nebo více hlav (pokud to dovolíme) uvnitř či na kraji úseku. Stav řídící jednotky. (Stav výstupních pásek není součástí konfigurace.)

Displej — Obsah políček pod hlavama vstupních a pracovních pásek, stav řídící jednotky.

Krok — Obsah políček, který bude zapsán pod hlavy pracovních a výstupních pásek, informace kam se pohnou jednotlivé hlavy *TS*, změna stavu řídící jednotky.

Přechodová funkce — Na základě displeje *TS* určí krok *TS* (mnohoznačné pro nedeterministické *TS*).

Výpočet — Provádění kroků vybíraných na základě displeje pomocí přechodové funkce.

Měl by být definován čas Turingova stroje a prostor Turingova stroje. (Prostor je určen pouze pracovníma páskama, nebo-li páskama, které jsou read/write!)

U nedeterministického TS je problém s interpretací výsledku výpočtu. Problém je způsoben možnou nejednoznačností výsledku. Proto necháváme nedeterministické TS odpovídat pouze ANO/NE, s tím, že v případě různých odpvědí má přednost odpověď ANO.

Poznámka 1.1 Tyto problémy s definicí jsou podobné problémům paralelních výpočtů. Mohli bychom například nechat TS vydat složitější výsledek za podmínky, že všechny větve výpočtu vydají tentýž výsledek. Mohli bychom definovat jako výsledek výpočtu „nejmenší číslo“ vydané nějakou větví výpočtu. („Největší číslo“ by nemuselo být definované).

Zvolené řešení přináší nejmenší množství problémů.

U nedeterministického TS je čas/prostor odhadnut minimálním nutným časem/prostorem, který umožňuje správný závěr.

(Pokud je výsledek ANO, stačí čas/prostor nejnenáročnější větve výpočtu vedoucí k ANO. Pokud je výsledek NE, je potřeba čas/prostor nejnáročnější větve výpočtu.)

1.2 Porovnávání funkcí

Následující symbolika je používána k vyjádření asymptotických porovnání funkcí. Symbol $\forall^*$ znamená až na konečně mnoho vyjímek (od určitého n_0). Symbol $\exists^\infty$ znamená existuje nekonečně mnoho. (Tak jak se neguje výraz s $\forall$ pomocí $\exists$, tak se neguje výraz s $\forall^*$ pomocí $\exists^\infty$.)

$$g \in O(f) \Leftrightarrow \exists\ r > 0\ \forall^* n\ g(n) < rf(n)$$

$$g \in o(f) \Leftrightarrow \forall\ r > 0\ \forall^* n\ g(n) < rf(n)$$

$$g \in \Omega_\infty(f) \Leftrightarrow \exists\ r > 0\ \exists^\infty n\ g(n) > rf(n)$$

$$g \in \omega(f) \Leftrightarrow \forall\ r > 0\ \exists^\infty n\ g(n) > rf(n)$$

$$g \in \Theta(f) \Leftrightarrow \exists\ r_1, r_2 > 0\ \forall^* n\ r_1 f(n) \le g(n) \le r_2 f(n)$$

$$O(F) = \bigcup_{f \in F} O(f)$$

$$o(F) = \bigcap_{f \in F} o(f)$$

$$\Omega_\infty(F) = \bigcup_{f \in F} \Omega_\infty(f)$$

$$\omega(F) = \bigcap_{f \in F} \omega(f)$$

$$\Theta(F) = \bigcup_{f \in F} \Theta(f)$$

Množinu všech funkcí můžeme zapsat následovně:

$$O(F) \cup \omega(F) = O(F) \triangle \omega(F) = o(F) \cup \Omega_\infty(F) = o(F) \triangle \Omega_\infty(F)$$

2 Simulace, univerzální stroje (15. dubna 1998)

Věta 2.1 *(Věta o lineární kompresi) Nechť r je kladné celé číslo. Pro libovolný TS pracující v prostoru $s(n)$ můžeme sestrojit TS pracující v prostoru právě $\lceil s(n)/r \rceil$.*

Důkaz: Na každé pracovní pásce bude jedno políčko obsahovat r políček původní abecedy. □

Věta 2.2 *(Věta o lineárním zrychlení) Nechť r je kladné celé číslo. Pro libovolný TS pracující v čase $t(n)$ můžeme sestrojit TS pracující v čase právě $n + \lceil n/r \rceil + 6 \cdot \lceil t/r \rceil$.*

Důkaz: Okopírujeme vstupní pásky na pracovní pásky jejichž jedno políčko obsahuje r políček původní abecedy. Poté pracujeme na zahuštěných pracovních páskách. Nyní jsme schopni r kroků původního *TS* odsimulovat ze znalostí tří políček v okolí hlavy každé zahuštěné pásky. Jejich přečtení a modifikaci můžeme provést na 6 kroků (tzv. tanečkem). Čas $n + \lceil n/r \rceil$ potřebujeme na zkopírování vstupních pásek na pracovní pásky a na přesun na začátek pracovních pásek. (Nejhorší je případ jediné vstupní pásky.)
□

Poznámka 2.1 Je-li $s(n)$ prostor spotřebovaný původním *TS*, pak prostor spotřebovaný nově vytvořeným *TS* je nejvýš $\big(n + s(n)\big)/r + k$, kde k je počet vstupních a pracovních pásek, pomocí něhož můžeme odhadnout zaokrouhlovací chyby.

Důsledek 2.2.1 *Vzhledem k předchozí větě je pro čas $t > (1+\varepsilon)n$ ekvivalentní, zda je možno provést výpočet v čase t nebo v čase $O(t)$. Pro libovolné s je ekvivalentní, zda je možno výpočet provést v prostoru s nebo $O(s)$.*

Poznámka 2.2 Typem *TS* budeme rozumnět, zda se jedná o deterministický či nedeterministický *TS*. Později rozšíříme možnosti interpretace výsledku výpočtu *TS*.

Turingovy stroje různých typů se liší případnou víceznačností přechodové funkce. Typ *TS* rozhoduje, jak je interpretován výsledek výpočtu na základě výsledku výpočtu jednotlivých pokračování.

Přesněji je interpretace závislá na typu jednotlivých stavů *TS*. Typ *TS* určuje, jaké typy stavů se v průběhu výpočtu mohou vyskytovat.

Věta 2.3 *(1. o simulaci) Libovolný TS pracující v čase t a prostoru s je možno nahradit TS stejného typu s jednou pracovní páskou pracujícím v čase $O(t \cdot s)$ a prostoru $O(s)$.*

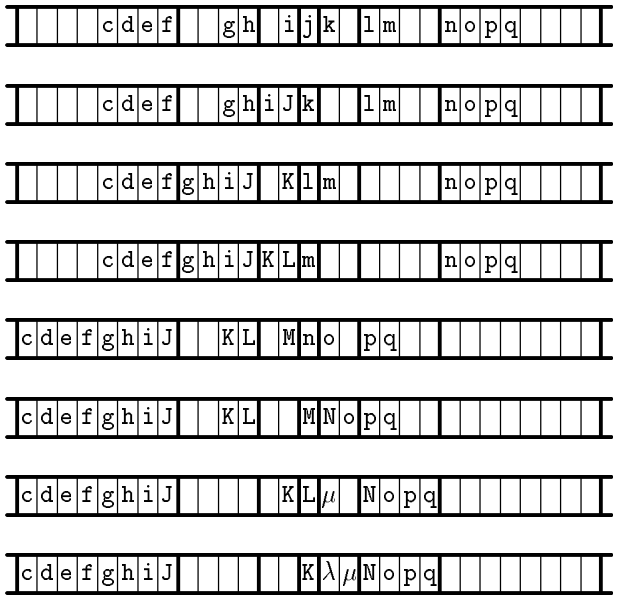
Důkaz: Do jednoho políčka pracovní pásky uložíme obsahy příslušného políčka každé pracovní pásky původního *TS*. Navíc si v políčku poznamenáme pro každou pásku, zda hlava (případně které hlavy) stojí na příslušném políčku. Abychom zjistili obsahy políček pod jednotlivými hlavami, musíme projet celou pásku. Totéž musíme provést při úpravě obsahu pásek a při posunu hlav. Na simulaci jednoho kroku nám stačí čas O(s). □

Poznámka 2.3 Věta umožňuje simulaci i pro stroje s více hlavama na jedné pásce.

Věta 2.4 *(2. o simulaci) Libovolný TSs jednou hlavou na každé pásce, pracující v čase t a prostoru s, je možno nahradit TS stejného typu s dvěma pracovníma páskama, pracujícím v čase $t' = O(t \cdot \log s)$ a prostoru $s' = O(s)$.*

Důkaz: Největší potíže působí fakt, že simulovaný *TS* může mít libovolně velký počet pracovních pásek. Nutně budeme muset na nějaké pásce simulovat větší počet pracovních pásek. Označme si tuto pásku p_0. Vzhledem k tomu, že k simulaci jednoho kroku *TS* potřebujeme znát obsahy políček pod hlavama na všech páskách, nemůžeme si dovolit kvůli simulaci jednoho kroku jezdit po pásce p_0 k políčkům, kde se hlavy nachází. (Potom by simulace jednoho kroku mohla trvat i s.) Strategie, která nám umožní simulaci v čase menším než $t \cdot s$, musí udržovat hlavu pásky p_0 „téměř“ na místě a posuny na páskách simulovat posunem obsahu pásky p_0 opačným směrem.

Popíšeme, jak je simulován posun hlavy na jedné pásce. Je-li pásek k, pak stejnou simulaci děláme $k-$krát na „$k-$krát tlustší pásce“.



Obr. 1: Simulace posunů na pásce p_0

Pásku p_0 rozdělíme na „bloky“. Blok B_i ($i \in \mathbb{Z}$) může obsahovat 0, $2^{|i|-1}$ nebo $2^{|i|}$ symbolů simulované pásky. Vždy bloky B_i a B_{-i} obsahují dohromady $2^{|i|}$ prvků. Dále je dodržováno pravidlo, že v blocích s menšími čísly jsou uchovávány levější symboly pásky.

Pro udržování hranic (a „půlhranic“) bloků je potřeba přidat k pásce p_0 „synchronizační stopu“. Značky synchronizační stopy je možno bez časových ztrát dynamicky doplňovat vždy, když se pracuje s nově nejvzdálenějším blokem od bloku B_0.

Druhou pásku používáme jako pomocný prostor k tomu, abychom se kvůli přesunům bloků na pásce p_0 nemuseli vracet.

Při odhadu celkového času simulace jedné pásky si stačí uvědomit, že práce s blokem $B_{\pm i}$ vynucuje pravidelné zaplnění ($2^{|j|-1}$ prvků) bloků $-|i|+1,\ldots,|i|-1$. Proto s blokem $B_{\pm i}$ pracujeme nejvýš jednou za $2^{|i|-1}$ kroků simulace. Tento krok simulace potom trvá s pomocnou páskou čas $2^{|i|+2}$. Celkový čas simulace můžeme poté odhadnout součtem přes všechny bloky a všech k pásek. Dostáváme tak odhad

$$t' \le k \sum_{i=0}^{1+\lfloor \log s \rfloor} 2^{|i|+2} \cdot \left\lfloor \frac{t}{2^{|i|-1}} \right\rfloor \le k \sum_{i=0}^{1+\lfloor \log s \rfloor} 8t = O(t \log s).$$

□

Poznámka 2.4 Vstupní/výstupní abeceda není podstatná z hlediska simulace, ale simulující stroj je musí mít stejné jako stroj simulovaný. Stejně tak není podstatný počet vstupních/výstupních pásek. V přesné formulaci věty o universálním *TS* by měly být uvedeny i tyto skutečnosti. V následující větě je tato skutečnost souhrně nazvána charakterem vstupu/výstupu.

Universálnímu stroji je nutno předat „program" (přechodovou funkci) simulovaného stroje. Tento program je možno předat na přídavé vstupní pásce. Potom ale universální stroj nebude stejného charakteru vstupu/výstupu (a nebudeme moci simulovat universální stroj tímtéž universálním strojem). Jinou možností je vybrat vstupní pásku s aspoň dvouznakovou abecedou a zakódovat na ní se vstupem simulovaného stroje zároveň program simulovaného stroje (zdvojíme symboly a jako oddělovač popisu stroje a jeho vstupu použijeme dvojici různých symbolů). To, že existuje vstupní páska s aspoň dvouznakovou abecedou nazveme netriviálním charakterem vstupu/výstupu.

Věta 2.5 *(1. o univerzálních TS) Pro libovolný netriviální charakter vstupu/výstupu a libovolný typ TS existuje „univerzální" TS daného charakteru vstupu/výstupu a typu, umožňující simulovat průběh výpočtu libovolného TS daného charakteru vstupu/výstupu a typu. Pro vztah mezi časovými nároky t (prostorovými nároky s) univerzálního U a simulovaného S TS platí:* $s_U(n) \le c_S^s \cdot (s_S(n)+1)$, $t_U(n) \le c_S^t t_S(n) \cdot (t_S(n)+c_S^t)$, *kde* $c_S^s \le 4|S|$ *a* $c_S^t \le 4|S|$ *pro* $|S|$ *značící délku zápisu programu simulovaného stroje.*

Věta 2.6 *(2. o univerzálních TS) Pro libovolný charakter vstupu/výstupu a libovolný typ TS existuje „univerzální" TS daného charakteru vstupu/výstupu a typu, umožňující simulovat průběh výpočtu libovolného TS daného charakteru vstupu/výstupu a typu s jednou hlavou na každé pásce. Pro vztah mezi časovými nároky t (prostorovými nároky s) univerzálního U a simulovaného S TS platí:* $s_U(n) \le c_S^s \cdot (s_S(n)+1)$, $t_U(n) \le c_S^t \cdot t_S(n)(\log t_S(n)+2)$, *kde* $c_S^s \le 4|S|$ *a* $c_S^t \le 8|S|^2$, *kde* $|S|$ *značící délku zápisu programu simulovaného stroje.*

Důkaz: (obou vět) Univerzální *TS* musí umět simulovat libovolný *TS*.

Problémy více pásek vyřešíme jako v předchozích větách.

Všímavý čtenář mne v tuto chvíli upozorní, že šířka pásky p_0 je konstantní, a my bychom chtěli, aby byla široká k, pro libovolný počet k pracovních pásek. Toto vyřešíme společně s problémem, že abecedy pracovních pásek simulovaných *TS* můžou být libovolně veliké. — Řešením je kódovat symboly pásky p_0 v dvojkové soustavě. Je-li s' počet políček nejvíce popsané pásky a u počet políček pásky p_0 nutných k zakódování jednoho políčka všech pásek (plus k zakódování přítomnosti jednotlivých hlav na políčku v případě simulace podle 1. věty o simulaci resp. k zakódování synchronizační stopy v případě simulace podle 2. věty), pak je potřeba $2s'u$ (resp. $4s'u$) políček pásky p_0 při simulaci podle 1. (resp. 2.) věty o simulaci. Uvědomme si, že k popisu přechodové funkce simulovaného stroje potřebujeme zápis obsahující zakódování několika displejů a kroků simulovaného stroje. K zakódování jednoho displeje a kroku potřebujeme dvakrát zakódovat symbol každé pracovní pásky tedy více než u bitů. Odtud $u < |S|$.

Posledním problémem je, že simulovaný *TS* může mít libovolně velkou množinu vnitřních stavů a libovolnou přechodovou funkci. — To nám nebude vadit, použijeme pásku reprezentující přechodovou funkci a pásku, na níž bude zaznamenán aktuální stav simulovaného *TS*. Tím se nám potřebný prostor zvětší pouze o konstantu nejvýš $2 \cdot |S|$.

Při odhadu konstanty c_S^t musíme postupovat opatrně. Simulaci každého kroku stroje S můžeme rozdělit do tří částí.

1. Zapsání displeje na pomocnou pásku.
2. Volba pokračování na základě přechodové funkce.
3. Zápisy na pásky a posuny hlav.

První a třetí krok provedeme obdobně jako v příslušné větě o simulaci. Druhý krok spočívá ve vyhledání displeje v přechodové funkci (v seznamu dvojic (displej,krok)). Při vhodném kódování přechodové funkce je možno tento krok provést v čase $4|S|$. Uvědomme si, že při simulaci nedeterministického stroje můžeme o použití vhodného kroku nedeterministicky rozhodnout. Odhad času 1. a 3. kroků je proveden ve větách o simulaci. Nyní jej upřesníme.

Při použití první věty o simulaci potřebujeme k opsání displeje čas nejvýš $2s'u$ (délka pásky p_0). K zápisu potřebujeme navíc čas pro posuny q pracovních hlav. Pro posuny $q < |S|$ pracovních hlav v průběhu zápisu nového obsahu políček pod hlavama stačí navíc čas $2qu$. Na 1. a 3. kroky simulace tedy potřebujeme celkem nejvýš $(4s'u + 2qu) \cdot t_S$ kroků. Po přičtení času za 2. kroky dostáváme $(4s'u + 2qu + 4|S|)t_S$. Po úpravě (díky $s' < t_S$) dostáváme čas $(4s'u + 2qu + 4|S|)t_S \le 4|S|t_S^2 + (2|S|^2 + 4|S|)t_S < 4|S|t_S(t_S + 4|S|)$.

Při použití druhé věty o simulaci opíšeme obsah políček pod jednotlivýma hlavama v čase $2u$, protože jej udržujeme na jednom místě pásky p_0. Na 1. a 3. kroky simulace potřebujeme celkem nejvýš $\big(2u + 8ku(1 + \lfloor \log s' \rfloor)\big) \cdot t_S$. Po přičtení času za 2. kroky dostáváme $(4|S| + 2u + 8ku(1 + \lfloor \log s' \rfloor))t_S$. Po úpravě díky $s' < t_S$ a $u, k < |S|$ dostáváme čas nejvýš $(6|S| + 8|S|^2(1 + \lfloor \log t_S \rfloor))t_S < 8|S|^2 t_S(\log t_S + 2)$. □

Poznámka 2.5 Komprimací pracovních pásek jsme ve větách o universálních strojích schopni snížit multiplikativní konstanty. Tak získáme odhad $t_U(n) \le |S|(t_S(n) + |S|)^2$, $s_U(n) \le |S|(s_S(n)+1)$ pro stroje s více hlavama na jedné pásce. Obdobně získáme odhad $t_U(n) \le |S|^2 t_S(\log t_S + 1)$, $s_U(n) \le |S|(s_S(n)+1)$ pro simulaci strojů s jednou hlavou na každé pásce.

Věta 2.7 *(O nedeterministické časové simulaci) Každý NTS (později $\boxed{\exists}$-TS nebo $\boxed{\forall}$-TS) pracující v čase t je možno nahradit TS stejného typu s dvěma pracovníma páskama pracujícím v čase $O(t)$. (Nový stroj bude přijímat tentýž jazyk).*

Důkaz: Nový stroj na první pásce uhodne t displejů a kroků původního stroje. K tomu potřebuje čas (a prostor) $O(t)$. Poté nový stroj s pomocí druhé pásky postupně pro jednotlivé pásky (deterministicky) kontroluje, zda vždy displej odpovídá dosud provedeným krokům původního stroje. (Na druhé pásce potřebuje mít nový stroj tolik hlav, kolik je maximálně hlav na jedné pásce původního stroje.) Pro každou pásku toto ověří v čase $O(t)$. Vzhledem ke konstantnímu (na vstupu nezávislém) počtu pásek původního stroje je celkový čas ověření toho, že byl uhodnut výpočet TS roven $O(t)$. Jde-li o $\boxed{\exists}$-TS, pak zamítáme, pokud nebyl uhodnut výpočet. Jde-li o $\boxed{\forall}$-TS, pak přijímáme, pokud nebyl uhodnut výpočet. Pokud byl uhodnut výpočet, přijímáme, pokud je poslední display přijímací. □

Poznámka 2.6 Pro $\boxed{\exists}$-TS a $\boxed{\forall}$-TS je možno pomocí předchozí věty vytvořit universální TS pracující v čase $t_U(n) \leq |S|^2 \cdot (t_S(n)+1)$. Zároveň tím ztratíme odhad prostorové složitosti.

3 Konstruovatelnost funkcí (29. června 1997)

Definice 3.1 Funkce $f : \mathbb{N} \to \mathbb{N}$ je časově konstruovatelná, pokud existuje (deterministický!) *TS*, který se pro každý vstup délky n zastaví v čase $f(n)$.

Definice 3.2 Funkce $f : \mathbb{N} \to \mathbb{N}$ je prostorově konstruovatelná, pokud existuje (deterministický!) *TS*, který pro každý vstup délky n použije právě prostor $f(n)$.

Věta 3.1 *Jsou-li $f_1 + f_2$, f_2 časově konstruovatelné funkce a platí-li*

$$f_1(n) > n \ \wedge \ \exists \varepsilon > 0 \ \forall^* n \ f_1(n) \geq \varepsilon f_2(n) + (1+\varepsilon)n,$$

potom je i f_1 časově konstruovatelná.

Důkaz: Nechť M_1 je *TS* pracující v čase právě $f_1 + f_2$. Nechť M_2 je *TS* pracující v čase právě f_2.

Budeme konstruovat stroj M_r, který by měl pracovat v čase právě f_1. Naší snahou je „odečíst" čas f_2. Jediné, čeho můžeme použít, je věta o zrychlení. Urychlíme výpočet M_1. V zrychleném režimu M_1 odsimulujeme jen tolik kroků, abychom výpočet urychlili o f_2. Poté již necháme M_1 doběhnout nezrychleně.

Nechť $r > 6$ je vhodné přirozené číslo. (Volbou r se budeme zabývat později.) V první fázi M_r zahustí vstup na tři pásky. Na první pásce bude urychlován *TS* M_1, na druhé bude urychlován *TS* M_2 a třetí páska slouží k odstranění počátečního zpoždění M_1.

Na první pásku je vstup zahuštěn r-krát, na druhou a třetí pásku $(r-6)$-krát. V průběhu první fáze počítá M_r modulo $r-6$, pokud fáze nekončí v čase dělitelném $r-6$, provede M_r několik (nejvýš $r-5$) kroků tak, aby fáze skončila v čase dělitelném $r-6$.

Čas první fáze je tedy přesně

$$(r-6) \cdot \left\lceil \frac{n}{r-6} \right\rceil.$$

V druhé fázi je M_1 urychlován na první pásce po dobu $6 \cdot (\lceil n/(r-6) \rceil + 1)$ kroků. Počet kroků je kontrolován posuny na třetí pásce. Za tuto dobu vykoná urychlený M_1 $r \cdot (\lceil n/(r-6) \rceil + 1)$ kroků.

Čas prvních dvou fází je přesně

$$(r-6) \cdot \left\lceil \frac{n}{r-6} \right\rceil + 6 \cdot \left\lceil \frac{n}{r-6} \right\rceil + 6 = r \cdot \left\lceil \frac{n}{r-6} \right\rceil + 6.$$

Dosud bylo odsimulováno $r \cdot \lceil n/(r-6) \rceil + r$ kroků M_1.

Ve třetí fázi počkáme $r-6$ kroků. Tím jsme dosáhli toho, že v průběhu prvních tří fází jsme odsimulovali stejný počet kroků M_1, jaký jsme spotřebovali čas.

Ve čtvrté fázi simulujeme M_2 na druhé pásce a souběžně pokračujeme v simulaci M_1 na pásce první. Stroj M_2 se nemusí zastavit přesně „na konci tanečku". Proto je stroj M_r informován o tom, kolik kroků ($k < r-6$) stroje M_2 chybělo k dokončení tanečku.

Vyčkáním $k = (r-6)\lceil f_2(n)/(r-6) \rceil - f_2(n)$ kroků zakončíme čtvrtou fázi.

Doba trvání čtvrté fáze je $6\lceil f_2(n)/(r-6) \rceil + k$. V průběhu čtvrté fáze je provedeno $r\lceil f_2(n)/(r-6) \rceil$ kroků M_1. Stroj M_1 je tedy ve čtvrté fázi urychlen o

$$(r-6)\left\lceil \frac{f_2(n)}{r-6} \right\rceil - k = f_2(n)$$

kroků.

V páté fázi je na první pásce stroj M_1 dosimulován již nezrychlovaně. Celkový čas výpočtu stroje M_r je tedy $(f_1 + f_2) - f_2 = f_1$.

Ještě je třeba ověřit, že je možno zvolit takové r, aby simulace stroje M_1 neskončila v průběhu prvních čtyř fází.

K tomu je potřeba, aby

$$(f_1 + f_2)(n) > r\left(\left\lceil \frac{n}{r-6} \right\rceil + 1\right) + r \cdot \left\lceil \frac{f_2(n)}{r-6} \right\rceil.$$

Odhadněme

$$r\left(\left\lceil \frac{n}{r-6} \right\rceil + 1\right) + r \cdot \left\lceil \frac{f_2(n)}{r-6} \right\rceil \leq 3r + \frac{r}{r-6}(f_2(n) + n).$$

Z předpokladu věty pro vhodné $\varepsilon > 0$ platí

$$\forall^* n \ f_1(n) \geq \varepsilon f_2(n) + (1+\varepsilon)n.$$

Zvolíme-li r tak, aby $r/(r-6) < 1 + \varepsilon$, pak

$$\forall^* n \ 3r + \frac{r}{r-6}(f_2(n) + n) < (1+\varepsilon)(f_2(n) + n).$$

Odtud dostáváme požadovaný odhad

$$\forall^* n \ (f_1 + f_2)(n) \geq (1+\varepsilon)(f_2(n) + n) >$$

$$> 3r + \frac{r}{r-6}(f_2(n) + n) \geq r\left(\left\lceil \frac{n}{r-6} \right\rceil + 1\right) + r \cdot \left\lceil \frac{f_2(n)}{r-6} \right\rceil.$$

Konečně mnoho výjimek je ošetřeno zvlášť. □

Věta 3.2 *Pokud $\exists \varepsilon > 0 \forall^* n \ f(n) \geq (1+\varepsilon)n$, pak f je časově konstruovatelná, právě když je vyčíslitelná v čase $O(f)$.*

Důkaz: Je-li f časově konstruovatelná, pak „ji" můžeme použít jako časové počítadlo a psát 1 na výstupní pásku, dokud počítadlo počítá.

Naopak, označíme $g(n)$ přesný čas *TS* M, který funkci f počítá v čase $O(f(n))$. (Tedy $\forall^* n \ g(n) < cf(n)$ pro vhodné c.) Časová konstruovatelnost funkce f plyne z věty 3.1. Funkce g je totiž časem *TS* M, funkce $f + g$ je časem *TS*, který „po dopočítání stroje M spočítá počet 1 na pásce".

Ukážeme ještě existenci vhodného ε. Nechť ε_1, ε_2 a ε_3 jsou kladná reálná čísla taková, že

$$\forall^* n \ f(n) \geq (1+\varepsilon_1) \cdot n$$

$$(1-\varepsilon_2)(1+\varepsilon_1) > 1$$

$$\varepsilon_3 = \min\{\varepsilon_2/c, (1-\varepsilon_2)(1+\varepsilon_1) - 1\}$$

Potom

$$\forall^* n \ f(n) = \varepsilon_2 \cdot f(n) + (1-\varepsilon_2) \cdot f(n) >$$

$$> (\varepsilon_2/c) \cdot g(n) + (1-\varepsilon_2)(1+\varepsilon_1)n \geq \varepsilon_3 \cdot g(n) + (1+\varepsilon_3)n.$$

□

Příklad 3.1 Funkce n^k, $n!$, c^n, 2^{n^k}, $n\lceil \log n \rceil^k$, $n \log^* n$ jsou časově konstruovatelné.

Věta 3.3 *Funkce f je prostorově konstruovatelná, právě když je vyčíslitelná v prostoru $O(f)$.*

Důkaz: Stačí použít větu o kompresi. □

Důsledek 3.3.1 *Každá časově konstruovatelná funkce je prostorově konstruovatelná.*

Příklad 3.2 Funkce $\lceil \log n \rceil$ je prostorově konstruovatelná.

Věta 3.4 *Je-li f rekurzívní funkce, pak existuje časově konstruovatelná funkce g, která je všude větší než f (neboli $\forall n\ g(n) > f(n)$).*

Důkaz: Nechť M je TS, který pro vstup 1^n dává výstup $1^{f(n)}$. Nechť M' je TS, který pro vstup x dává výstup $1^{|x|}$. Nechť M'' je TS vzniklý spojením M' a M. (M pracuje na výstupu M'.)

Označme $g(n)$ čas stroje M'' strávený na vstupu délky n. Z konstrukce g vyplývá, že g je časově konstruovatelná a větší než f. □

4 Základní věty o třídách časově/prostorově omezených výpočtů (15. dubna 1998)

Omezíme se v dalším na Turingovy stroje, jejichž hlavy na vstupních páskách nemohou „přejet“ první symbol blank. Jinak by možné polohy hlav na vstupních páskách ovlivnily počet různých konfigurací *TS*. Dále se omezíme na Turingovy stroje s jednou hlavou na každé pásce. Bez omezení by se třídy $\boldsymbol{DSpace}(s)$ a $\boldsymbol{NSpace}(s)$ nezměnily, ale mohly by být jiné třídy $\boldsymbol{DTime}(t)$ a $\boldsymbol{NTime}(t)$.

Definice 4.1 $\boldsymbol{DTime}(t)$, $\boldsymbol{DSpace}(s)$, $\boldsymbol{NTime}(t)$, $\boldsymbol{NSpace}(s)$.

Věta 4.1 *Počet různých konfigurací které mohou být dosaženy v průběhu výpočtu TS v prostoru $s(n) \geq \log n$ je nejvýš $2^{O(s)}$.*

Poznámka 4.1 Stav výstupních pásek není součástí konfigurace, stav vstupních pásek se až na polohy hlav nemění.

Důkaz: Dosažitelných konfigurací je nejvýš

$$(n_1+1)\cdot(n_2+1)\cdots(n_i+1)\cdot|\Sigma|^{ks(n)}\cdot|Q|\cdot s(n)^k,$$

kde $n = n_1+n_2+\cdots+n_i$ je délka vstupu (n_j jsou délky vstupu jednotlivých vstupních pásek), Σ je abeceda pásek, k je počet pracovních pásek a Q je množina vnitřních stavů. Výraz můžeme popsat tvarem

$$n^{O(1)}\cdot O(1)^{s(n)}\cdot s(n)^{O(1)}\cdot O(1) = n^{O(1)}\cdot O(1)^{s(n)}.$$

Za předpokladu $s(n) \geq \log n$ pak dostáváme $n^{O(1)}\cdot$ $\cdot O(1)^{s(n)} = O(1)^{s(n)} = 2^{O(s(n))}$. □

Poznámka 4.2 Je-li $|\delta|$ délka popisu přechodové funkce stroje M překódovaného do binární abecedy, pak počet možných konfigurací v prostoru $s(n)$ je nejvýš $2^{|\delta|(s(n)+\log n+\log s(n)+1)}$.

Věta 4.2 *Ke každému TS pracujícímu v prostoru $s(n) \geq$ $\geq \log n$, kde s je prostorově konstruovatená funkce, existuje TS pracující ve stejném prostoru, který se vždy zastaví.*

Důkaz: Daný *TS* může mít nejvýš $2^{O(s(n))}$ různých konfigurací. Stačí přidat počítadlo do $2^{O(s(n))}$ a při dosažení počítadla výpočet zarazit. (Počítadlo potřebuje prostor $O(s(n))$, použijeme větu o kompresi.) □

Poznámka 4.3 Předchozí věta platí i v případě, kdy máme zajištěno, že *TS* pracuje v omezeném prostoru $s \geq \log n$. Přesnou velikost prostoru předem znát nemusíme. (Zda se *TS* zacyklí, testujeme pomocí časového počítadla velikosti, kterou zvětšujeme s tím, kolik prostoru *TS* doposud spotřeboval.)

Poznámka 4.4 Podle předchozí poznámky pro prostorově konstruovatelnou funkci $s \geq \log n$ existuje deterministický *TS*, který spotřebuje prostor s a zastaví se.

Věta 4.3 *Je-li $t(n) > n$ časově konstruovatelná a $s(n) \geq$ $\geq \log(n)$ prostorově konstruovatelná, potom*

$$\boldsymbol{D}\ldots(f) \subseteq \boldsymbol{N}\ldots(f)$$

$$\boldsymbol{NTime}(t) \subseteq \boldsymbol{DSpace}(t)$$

$$\boldsymbol{NSpace}(s) \subseteq \boldsymbol{DTime}(2^{O(s)})$$

Důkaz: První vztah je triviální vzhledem k tomu, že deterministický *TS* můžeme chápat jako speciální případ nedeterministického *TS*.

K důkazu platnosti druhého vztahu si stačí uvědomit, že „víceznačnou“ přechodovou funkci *TS* můžeme nahradit funkcí nejvýš dvojznačnou (případně právě dvojznačnou). Samozřejmě kvůli tomu musíme zvětšit stavový prostor *TS*. Výpočetní čas se také konstanta–krát prodlouží. Nyní stačí přidat pásku délky $O(t)$ obsahující nuly a jedničky. Simulující *TS* se v každém taktu posune po přidané pásce a podle symbolu na pásce vybere příslušnou větev výpočtu. Stačí nám tedy prostor $O(t)$. Podle věty o kompresi stačí t.

Možných konfigurací *TS* pracujícího v prostoru s je $2^{O(s)}$. Daný *TS* můžeme upravit tak, aby měl jednoznačnou koncovou konfiguraci a aby stále ještě pracoval v prostoru s. Nejprve popíšeme princip simulace v čase $2^{O(s)}$. Simulace ve skutečnosti zjišťuje, zda existuje průchod grafem možných konfigurací od počáteční ke koncové konfiguraci. Graf konfigurací *TS* obsahuje konfigurace jako vrcholy. Hrany vedou z konfigurace α do konfigurace β, právě když přechodová funkce *TS* umožňuje v jednom kroku přejít z konfigurace α do konfigurace β. Uvědomte si, že stupně vrcholů grafu jsou omezeny konstantou. (Konfigurace β se od konfigurace α může lišit pouze stavem a lokálním okolím hlav.) Pro simulaci si opět přidáme pomocnou pásku, sloužící k označování již dosažených vrcholů. Na tuto pásku zapíšeme 1 na k-té políčko v případě, že k-tá konfigurace je dosažitelná z počáteční konfigurace. „Konfiguraci budeme chápat jako binární zápis svého čísla“. Použijeme-li na graf konfigurací prohledávání do hloubky, musíme ještě odsimulovat zásobník vrcholů (na další přidané pásce). Jedna operace se „zásobníkem“ nás bude stát čas $O(s)$. Zjištění, zda byl vrchol již zpracován, nás stojí čas $2^{O(s)}$. Pro celkový čas průchodu grafem konfigurací do hloubky dostáváme čas $O(N\cdot O(s)+$ $+M\cdot 2^{O(s)})$, kde N resp. M je počet vrcholů resp. hran grafu konfigurací. Po dosazení $N = 2^{O(s)}$, $M = 2^{O(s)}$ dostáváme

$$N\cdot O(s) + M\cdot 2^{O(s)} = 2^{O(s)}\cdot O(s) + 2^{O(s)}\cdot 2^{O(s)} = 2^{O(s)}.$$

□

Poznámka 4.5 Při důkazu druhého tvrzení jsme mohli využít přidané pásky s dostatečně velkou abecedou, abychom pokryli „libovolně velký“ nedeterminismus daného *TS*. Tím bychom vynechali fázi převodu na *TS* s dvojznačnou přechodovou funkcí.

Věta 4.4 *(Savitch) Je-li $s(n) \geq \log n$ prostorově konstruovatelná funkce, pak*

$$\boldsymbol{NSpace}(s) \subseteq \boldsymbol{DSpace}(s^2)$$

Důkaz: K důkazu Savitchovy věty použijeme opět odhad počtu možných konfigurací *TS*. Tento počet ($p = 2^{O(s)}$) je horním odhadem času nejrychlejšího výpočtu.

Popis algoritmu se asi zjednodušší, zavedeme-li proceduru (funkci) $Access\_2^k(\alpha,\beta)$, která vydá `true`, právě když je konfigurace β dosažitelná z konfigurace α v nejvýš 2^k krocích.

```
procedure Access(k, α, β);
var γ : Konfigurace;
begin
  if k = 0 then Access = (α = β) or Next(α, β)
  else
      begin
        Access := false;
        for γ ∈ Konfigurace do
            if Access(k − 1, α, γ) and
            Access(k − 1, γ, β) then
                Access := true
      end
end
```

Alg. 1: Funkce $Access\_2^k$

Velikost lokálních proměnných funkce je $O(s)$ a funkce $Access\_2^k$ $(k > 0)$ volá pouze funkci $Access\_2^{k-1}$. Funkce $Access\_2^0$ žádné funkce nevolá.

Prostor potřebný pro funkci $Access\_2^k$ včetně rekurentně volaných funkcí je tedy $k \cdot O(s)$. Proto $Access\_2^{\lceil \log p \rceil}$ vyžaduje prostor $O(s) \cdot O(s) = O(s^2)$.

Na závěr je třeba si uvědomit, že při simulaci na TS si vystačíme se stejným prostorem. □

Poznámka 4.6 Předpoklad $s(n) \geq \log n$ byl využit k odhadu počtu různých konfigurací. Konstruovatelnost je potřeba k tomu, abychom mohli „naalokovat místo pro proměnné“.

5 Věty o hierarchii (29. září 1997)

Věta 5.1 *Jsou-li s, s' prostorově konstruovatelné funkce, $s' \in \omega(s)$ a $s'(n) \geq \log n$, pak existuje jazyk v třídě*

$$\boldsymbol{DSpace}(s') \setminus \boldsymbol{DSpace}(s)$$

Důkaz: K důkazu je použita tzv. diagonální metoda. Definujeme jazyk L následovně: $1^k0x \in L$ právě když x kóduje DTS, DTS_x nepřijme 1^k0x v prostoru s a universální TS provede simulaci v prostoru s'.

Nejprve ukážeme, že L je z $\boldsymbol{DSpace}(s')$. Ze vstupu 1^k0x jsme schopni vygenerovat výstup $(1^k0x|x)$ s konstantním pracovním prostorem. Universální TS provede simulaci na vstupu $(1^k0x|x)$ v prostoru s'. K dokončení první části důkazu potřebujeme trik (který budeme později používat u transformací v logaritmickém prostoru).

Problém je v tom, že může být $|x| > s'(|1^k0x|)$, takže nemáme dost pracovního prostoru, abychom zaznamenali celý vstup universálního TS. Místo toho si pamatujeme pouze polohy hlav na vstupních páskách universálního TS. Pokaždé, kdy potřebujeme znát obsah políčka vstupní pásky universálního TS, spustíme znovu algoritmus generující vstup a čekáme, až bude vygenerováno příslušné políčko. Ostatní symboly vstupní pásky nás v tuto chvíli nezajímají. (Potřebujeme k tomu pomocné počítadlo až do délky vstupní pásky.)

Celkem nám k simulaci universálního TS stačí pracovní prostor $O(1) + 2\log(|1^k0x|) + 2\log(|x|) + s' = O(s'(n))$. Podle věty o kompresi nám stačí prostor $s'(n)$.

V druhé části důkazu ukážeme, že L není z $\boldsymbol{DSpace}(s)$.

Nechť y je DTS rozpoznávající L v prostoru s. Podle věty o simulaci universální TS simuluje DTS_y v prostoru $c_y \cdot s$. Protože $s' \in \omega(s)$, existuje $n > 1 + |y|$, pro nějž $c_y \cdot s < s'$.

Pro slovo $1^{n-|y|-1}0y$ má universální TS dost prostoru na to, aby provedl celou simulaci. Algoritmus tedy slovo $1^{n-|y|-1}0y$ přijme, právě když jej TS_y nepřijme. TS_y tedy nepřijímá jazyk L. □

Věta 5.2 *Jsou-li t, t' časově konstruovatelné funkce a $t' \in \in \omega(t \log t)$ a $\exists\, \varepsilon > 0\ t' \geq (1+\varepsilon)n$, pak existuje jazyk v třídě*

$$\boldsymbol{DTime}(t') \setminus \boldsymbol{DTime}(t)$$

Důkaz: K důkazu je opět použita diagonální metoda. Definujeme jazyk L následovně: $1^k0x \in L$ právě když x kóduje DTS, DTS_x nepřijme 1^k0x v čase t a universální TS provede simulaci v čase t'.

Důkaz příslušnosti L do $\boldsymbol{DTime}(t')$ je nyní jednodušší. V čase $O(|1^k0x|)$ připravíme vstup $(1^k0x|x)$ universálnímu TS a spustíme jej. Čas spotřebovaný algoritmem je $O(|1^k0x|) + t' = O(t')$, a protože $t' > (1+\varepsilon)n$, stačí nám podle věty o zrychlení čas t'.

Důkaz druhé části je obdobný jako v předchozí větě. Jediný rozdíl je v tom, že pracuje-li TS_y v čase t, pak universální TS odsimuluje TS_y v čase $c_y \cdot t \log t$. □

Poznámka 5.1 Potřebovali jsme zde to, že na každé pásce je jediná hlava!

Věta 5.3 *Jsou-li s, s' prostorově konstruovatelné funkce, $s' \in \omega(s)$ a $s'(n) \geq \log n$, pak existuje jazyk v třídě*

$$\boldsymbol{NSpace}(s') \setminus \boldsymbol{NSpace}(s)$$

Důkaz: Důkaz by byl úplně stejný, jako důkaz hierarchické věty pro $\boldsymbol{DSpace}$, pokud bychom uměli univerzálním strojem v prostoru $c_x \cdot s_x(|y|)$ nedeterministicky testovat, zda NTS_x nepřijme vstup y. To je garantováno následující větou. □

Věta 5.4 *Pro prostorově konstruovatelnou funkci $s(n) \geq$ $\geq \log n$ existuje funkce co, kterou je možno vyčíslit deterministicky, s následující vlastností:*

Je-li L jazyk rozpoznávaný pomocí NTS_x v prostoru s, pak jazyk $co - L$ je rozpoznávaný pomocí $NTS_{co(x)}$ v prostoru s. ($x \in L \overset{def}{\Leftrightarrow} x \notin co - L$)

Důkaz: $NTS_{co(x)}$ bude pracovat ve fázích. V i-té fázi spočítá počet $p(i)$ dosažitelných konfigurací stroje NTS_x v čase nejvýš i (zároveň testuje zda je mezi nimi přijímací konfigurace). Pokud je mezi nimi přijímací konfigurace, výpočet stroje $NTS_{co(x)}$ končí zamítnutím. Ve chvíli, kdy $p(i) = p(i+1)$ a žádná (dosud) dosažitelná konfigurace není přijímací, stroj $NTS_{co(x)}$ přijme.

Zbývá ukázat, jak na základě $p(i)$ $NTS_{co(x)}$ zjistí, zda daná konfigurace α je či není dosažitelná v čase $i+1$. K tomu stačí projít všechny konfigurace β_j $(j = 1 \ldots p = 2^{O(s)})$ a $p(i)$–krát uhodnout, že konfigurace $\gamma_k = \beta_{j_k}$ $(k = 1 \ldots p(i) < p)$ je dosažitelná v čase nejvýš i. Pro každou takto uhodnutou konfiguraci γ_k je třeba nejvýš i–krát uhodnout předcházející konfiguraci δ_ℓ $(\ell = 1 \ldots i_k \leq i)$, abychom zkontrolovali, že konfigurace je skutečně dosažitelná. Pokud kontrola selže, tato větev výpočtu zamítá. Je-li nyní α dosažitelná v nejvýš jednom kroku z γ_k, je α dosažitelná v čase nejvýš i. Pokud není α dosažitelná v nejvýš jednom kroku z γ_k, pokračuje výpočet dalším j. Je-li na konci cyklu $k < p(i)$, tato větev výpočtu zamítá. Je-li na konci cyklu $k = p(i)$ a α nebyla dosažitelná v nejvýš jednom kroku z žádné z konfigurací γ., pak α není dosažitelná v čase nejvýš i. Uvedený test buď končí zamítnutím (nesprávně uhodnutá větev výpočtu), nebo odpovědí ANO, nebo odpovědí NE. Test nemůže odpovědět nesprávně.

$NTS_{co(x)}$ spočítá $p(i+1)$ průchodem všech konfigurací α_m $(m = 1, \ldots, p)$. Pro každou konfiguraci otestujeme na základě $p(i)$, zda je dosažitelná v čase $i+1$. Za každou odpověď ANO zvýší počítadlo (předtím však testujeme, zda se jedná o konfiguraci přijímací).

Prostorové nároky stroje $NTS_{co(x)}$ jsou následující: Je-li p počet konfigurací NTS_x, pak $i, p(i), p(i+1), j, k, \ell, m \leq p = 2^{O(s)}$. Zvolíme-li dostatečně velkou abecedu pracovních pásek, stačí nám na zakódování čísel i, $p(i)$ a $p(i+1)$ a konfigurací α_m, β_j, γ_k, δ_ℓ prostor s. Konstrukce stroje $NTS_{co(x)}$ byla popsána algoritmicky. □

Translační lemma

Translační lemma je založeno na principu anglicky zvaném padding. Tento princip bude popsán následující větou.

Definice 5.1 Pro funkci $f(n) > n$ a jazyk L definujeme jazyk L_f následovně:

$$L_f \stackrel{def}{=} \{1^k 0x \mid x \in L \wedge k + 1 + |x| = f(|x|)\}$$

(Slovo „odpovídající“ slovu x má délku $f(|x|)$).

Věta 5.5 *(Prodlužovací časová) Nechť $f(n)$ je časově konstruovatelná funkce větší než $n+1$ a $g(n)$ je časově konstruovatelná funkce větší než $(1+\varepsilon)n$ pro nějaké $\varepsilon > 0$. Pak $L \in \boldsymbol{DTime}(g(f(n))) \Leftrightarrow L_f \in \boldsymbol{DTime}(g(n))$ a obdobně $L \in \boldsymbol{NTime}(g(f(n))) \Leftrightarrow L_f \in \boldsymbol{NTime}(g(n))$.*

Věta 5.6 *(Prodlužovací prostorová) Nechť $f(n), g(n)$ jsou prostorově konstruovatelné funkce větší než n. Pak $L \in \boldsymbol{DSpace}(g(f(n))) \Leftrightarrow L_f \in \boldsymbol{DSpace}(g(n))$ a obdobně $L \in \boldsymbol{NSpace}(g(f(n))) \Leftrightarrow L_f \in \boldsymbol{NSpace}(g(n))$.*

Důkaz: Mějme algoritmus „s nároky“ $g(f(n))$ rozpoznávající L. Zda „$y \in L_f$?“ můžeme rozhodnout následujícím algoritmem: Nejprve zkontrolujeme, že y je tvaru $1^k 0x$. Dále zkontrolujeme, že $k + |x| + 1 = f(|x|)$. K tomu potřebujeme nárok velikosti $|y|$ (generujeme $f(|x|)$ a pokud výsledek přesáhne $|y|$, zamítáme). Poté zkontrolujeme s nárokem $g(f(|x|)) = g(|y|)$, že $x \in L$. Použitím věty o lineárním zrychlení resp. o lineární kompresi vytvoříme algoritmus s nároky $g(n)$ rozpoznávající L_f.

Mějme naopak algoritmus „s nároky“ $g(n)$ rozpoznávající L_f. Zda „$x \in L$?“ můžeme rozhodnout následujícím algoritmem: Nejprve vygenerujeme y ve tvaru $1^{f(|x|)-|x|-1}0x$. K tomu potřebujeme nárok velikosti $O(f(|x|))$. Poté zkontrolujeme, s nárokem $g(|y|) = g(f(|x|))$, že $y \in L_f$. Použitím věty o lineárním zrychlení resp. o lineární kompresi vytvoříme algoritmus s nároky $g(f(n))$ rozpoznávající L. □

Věta 5.7 *(Translační lemma – časová formulace) Nechť t_1, t_2, f jsou časově konstruovatelné funkce, nechť $f(n) > n+1$, a $\exists \varepsilon > 0\, t_1(n), t_2(n) > (1+\varepsilon)n$, pak*

$$\boldsymbol{DTime}(t_1(n)) \subseteq \boldsymbol{DTime}(t_2(n)) \Rightarrow$$

$$\Rightarrow \boldsymbol{DTime}(t_1(f(n))) \subseteq \boldsymbol{DTime}(t_2(f(n)))$$

a podobně

$$\boldsymbol{NTime}(t_1(n)) \subseteq \boldsymbol{NTime}(t_2(n)) \Rightarrow$$

$$\Rightarrow \boldsymbol{NTime}(t_1(f(n))) \subseteq \boldsymbol{NTime}(t_2(f(n)))$$

Důkaz: Nechť $A \in \boldsymbol{DTime}(t_1(f(n)))$. Podle prodlužovací věty je $A_f \in \boldsymbol{DTime}(t_1(n))$. Z předpokladu implikace pak $A_f \in \boldsymbol{DTime}(t_2(n))$. Opět z prodlužovací věty dostáváme $A \in \boldsymbol{DTime}(t_2(f(n)))$.

Stejně pro $\boldsymbol{NTime}$. □

Poznámka 5.2 Vzhledem k hierarchickým větám pro $\boldsymbol{DSpace}$ a $\boldsymbol{NSpace}$ ztrácí prostorová formulace translačního lemmatu na významu.

Stejný důkaz jako časová formulace translačního lemmatu má následující prostorová formulace:

Věta 5.8 *(Translační lemma – prostorová formulace) Nechť s_1, s_2, f jsou prostorově konstruovatelné funkce, nechť $f(n), s_1(n), s_2(n) > n$, pak*

$$\boldsymbol{DSpace}(s_1(n)) \subseteq \boldsymbol{DSpace}(s_2(n)) \Rightarrow$$

$$\Rightarrow \boldsymbol{DSpace}(s_1(f(n))) \subseteq \boldsymbol{DSpace}(s_2(f(n)))$$

a podobně

$$\boldsymbol{NSpace}(s_1(n)) \subseteq \boldsymbol{NSpace}(s_2(n)) \Rightarrow$$

$$\Rightarrow \boldsymbol{NSpace}(s_1(f(n))) \subseteq \boldsymbol{NSpace}(s_2(f(n)))$$

Silnější tvrzení dostaneme, pokud se nebudeme opírat o prodlužovací lemma. V předpokladech nepotřebujeme $s_1(n), s_2(n) > n$. Stačí $s_2(n) > \log n$ a $s_2(f(n))$ prostorově konstruovatelná.

Důkaz: Důkaz provedeme zároveň pro deterministickou i nedeterministickou formulaci. Typ stroje totiž nebude v důkazu hrát žádnou roli.

V důkazu použijeme místo jazyka L_f jazyk L_2 definovaný na základě stroje M_1 rozpoznávajícího L_1 v prostoru $s_1(f(n))$
$L_2 = \{x01^k \mid M_1$ přijme x v prostoru nejvýš $s_1(|x| + 1 + k)\}$ Stroj M_2 rozpoznávající L_2 vytvoříme následovně: Vytvoříme modifikaci M_1' stroje M_1, která má jedinou jednostrannou pracovní pásku, a pracuje v nejvýš tak velkém prostoru jako M_1. Pro y na vstupu M_2 nejprve označí $s_1(|y|)$ políček pásky (s_1 konstruovatelná), pak M_2 simuluje M_1' a přijme dané slovo, jen pokud M_1' přijal pouze s použitím označených políček.

Zřejmě M_2 pracuje v prostoru $s_1(n)$, takže L_2 je možno rozpoznat v prostoru $s_1(n)$ a podle předpokladu implikace také v prostoru $s_2(n)$. Nechť M_3 rozpoznává L_2 v prostoru $s_2(n)$.

Vytvoříme stroj M_4 rozpoznávající jazyk L_1 následovně: Použijeme modifikaci M_3' stroje M_3, takovou, že stroj M_3' se pro každý vstup zastaví. M_4 bude simulovat M_3' postupně na vstupech $x01^i$ pro $i = 0, 1, \ldots$. Kdykoli je čtecí hlava stroje M_3' uvnitř x, je hlava M_4 na odpovídajícím místě. Je-li však hlava M_3' napravo od x, M_4 používá počítadla k evidenci její polohy. Délka počítadla je nejvýš $1 + \log i$. Pokud M_3' přijme, pak M_4 přijme. Jinak M_4 zvětší i, pokud počítadlo nepřesáhne $s_2(f(n))$ políček pásky. Ve chvíli, kdy počítadlo přesáhne $s_2(f(n))$ políček pásky, M_4 odmítne.

Nyní jestliže M_4 přijal x, tak M_3 přijal pro nějaké i vstup $x01^i$, tedy $x01^i \in L_2$ a M_1 přijme x, tedy $x \in L_1$.

Jestliže naopak $x \in L_1$, pak $x01^i \in L_2$ pro $i = f(|x|) - |x| - 1$. Počítadlo na i potřebuje $\log(f(|x|) - |x| - 1) \leq s_2(f(|x|))$ prostoru. □

Cvičení 5.1 Dokažte, že $\boldsymbol{DTime}(2^n) \subsetneq \boldsymbol{DTime}(n2^n)$. (Použijte translační lemma pro $f = 2^n$ a $f = n + 2^n$.)

6 Na hranici vyčíslitelnosti (16. dubna 1998)

Definice 6.1 Funkce $f : \mathbb{N} \to \mathbb{N} \cup \{?\}$ je částečně rekurzivní, pokud existuje (deterministický) Turingův stroj M, který se pro vstup 1^n zastaví, právě když je $f(n) \in \mathbb{N}$. Pokud se M zastaví, pak vydá výstup $1^{f(n)}$.

Definice 6.2 Částečně rekurzivní funkce $f : \mathbb{N} \to \mathbb{N}$ je totální (rekurzivní).

Definice 6.3 Jazyk L je rekurzivně spočetný, pokud existuje (deterministický) Turingův stroj, který se pro vstup x zastaví, právě když $x \in L$.

Definice 6.4 Jazyk L je rekurzivní, pokud existuje (deterministický) Turingův stroj, který se vždy zastaví a pro vstup $x \in L$ vydá odpověď ANO.

Mějme universální stroj U simulující všechny deterministické Turingovy stroje (s jednou vstupní páskou nad abecedou {0,1}).

Nechť $\{TM_k\}_{k=0}^{\infty}$ je posloupnost Turingových strojů odpovídající vstupům k universálního stroje U. (Pro některá k nemusí být TM_k Turingův stroj, ovšem v posloupnosti se všechny Turingovy stroje vyskytují.)

Definice 6.5 Je-li TM_k deterministický Turingův stroj, označme $s_k(n) \in \mathbb{N} \cup \infty$ jeho prostor a L_k jazyk jím rozpoznávaný.

Lemma 6.1 *Existuje TSM, který pro vstup k, n, m rozhodne, zda TM_k je deterministický Turingův stroj a zda $s_k(n) \leq m$. Stroj M k rozhodnutí potřebuje prostor $s_M \leq$ $\leq n + (m + \log n + 1)\lceil \log k \rceil$.*

Důkaz: Vytvoříme stroj M' modifikací universálního stroje. Stoj M' nejprve zkontroluje, zda k kóduje přechodovou funkci, a poté pro každý vstup délky n provedeme simulaci v prostoru m, (přidáním „hodin" do počtu konfigurací stroje TM_k zajistí konečnost výpočtu). Na hodiny potřebujeme podle poznámky 4.2 prostor $\lceil \lceil \log k \rceil \cdot (m + \log n + \log m + 1) \rceil$. Na simulaci stačí prostor $\lceil \log k \rceil \cdot m$, navíc potřebujeme prostor n pro generování všech vstupů délky n.

Požadovaný stroj M vytvoříme ze stroje M' na základě věty o lineární kopresi. □

Lemma 6.2 *(O konečném počtu výjimek) Nechť L je libovolný jazyk. Pro každé k a n_0 existuje K tak, že je-li TM_k deterministický Turingův stroj, je i TM_K deterministický Turingův stroj, navíc pro $x > n_0$ platí $x \in$ $\in L_K \Leftrightarrow x \in L_k$ a pro $x \leq n_0$ platí $x \in L_K \Leftrightarrow x \in L$.*

Dále pro $n > n_0$ je $s_K(n) = s_k(n)$ a pro $n \leq n_0$ je $s_K(n) = 1$.

Důkaz: Pro libovolný konečný jazyk existuje konečný automat, který jej rozpoznává. Turingův stroj TM_K necháme pro vstupy delší než n_0 pracovat jako stroj TM_k. Pro vstupy délky nejvýš n_0 použijeme konečný automat rozpoznávající jazyk $L \cap \{0,1\}^{\leq n_0}$. K rozvětvení potřebujeme přidat do řídící jednotky počítadlo do n_0 a na začátku výpočtu ověřit, zda vstup je/není delší než n_0. (Pokud do zpotřebovaného protoru TS počítáme i stání na políčku blank bez snahy zapisovat, musíme vycházet z jednopáskového stroje přijímajícího jazyk L_k v prostoru s_k.) □

Věta 6.3 *(Borodinova o mezerách) Nechť $g(n) > n$ je totální rekurzivní funkce, pak existuje totální rekurzivní funkce $s(n)$ taková, že $\boldsymbol{DSpace}(s(n)) = \boldsymbol{DSpace}(g(s(n)))$.*

Důkaz: Pro každé m definujeme $s(m) \geq 1$ na základě $s_1(m), \ldots, s_m(m)$. Zajistíme, aby žádné z $s_k(m) \mid k = 1, \ldots, m$ nepadlo do intervalu $(s(m), g(s(m))\rangle$.

Je-li pak $L_k \in \boldsymbol{DSpace}(g(s(n)))$, pak, jelikož pro $n \geq k$ platí $s_k(n) \leq g(s(n)) \Leftrightarrow s_k(n) \leq s(n)$, je $s_k(n) \leq s(n)$ pro $n \geq k$. Podle lemmatu o konečném počtu výjimek existuje K tak, že $L_K = L_k$ a $s_K(n) \leq \begin{cases} s_k(n) \leq s(n) & \text{pro } n > k \\ 1 \leq s(n) & \text{pro } n \leq k \end{cases}$, tedy $L_k \in \boldsymbol{DSpace}(s(n))$.

Uvedenou podmínku zajistíme například následovně: Postupně volíme s z posloupnosti $1, g(1), g(g(1)), \ldots$ a vždy testujeme, zda $\forall k \leq m$ platí $(s_k(m) \leq s \Leftrightarrow s_k(m) \leq$ $\leq g(s))$. Když je podmínka poprvé splněna, zvolíme $s(m) =$ $= s$. Protože $g(n) > n$, každé k může zabránit splnění podmínky nejvýš jednou a podmínka je splněna pro některé $s \in \{g^{(i)}(1)\}_{i=0}^{m}$. □

Poznámka 6.1 (modifikovaná věta o mezerách) Je-li navíc zadána rekurzivní funkce f, můžeme v Borodinově větě o mezerách požadovat $s(n) \geq f(n)$.

Stačí totiž volit s z posloupnosti $f(n), g(f(n)), g(g(f(n))), \ldots$.

Důsledek 6.3.1 *Pokud v modifikované větě o mezerách $g(n) \notin O(n)$ je prostorově konstruovatelná, pak s při požadavku $s(n) > \log n$ není prostorově konstruovatelná.*

Důkaz: Pokud by byla $s(n)$ prostorově konstruovatelná, byla by i $g(s(n)) \notin O(s(n))$ prostorově konstruovatelná. Navíc $s(n), g(s(n)) > \log n$. Podle věty o hierarchii pak ale $\boldsymbol{DSpace}(s(n)) \neq \boldsymbol{DSpace}(g(s(n)))$. □

Věta 6.4 *(Blumova o zrychlení) Nechť r je totální rekurzivní funkce, pak existuje rekurzívní jazyk L takový, že $L_i =$ $= L \Rightarrow \exists j L_j = L \wedge \forall^* n r(s_j(n)) \leq s_i(n)$.*

Důkaz: Nejprve si uvědomme, že bez újmy na všeobecnosti můžeme předpokládat, že r je neklesající prostorově konstruovatelná funkce a $r(n) \geq n^2$. (Nechť $r'(n)$ je délka nejvíce popsané pásky při výpočtu $r(1), r(2), \ldots, r(n), n^2$.)

Definujme funkci h následovně:

$$h(1) = 2, \quad h(n) = r(h(n-1)).$$

(Všimněme si, že $h(n) \geq 2^{2^{n-1}}$.)

Pokud se nám podaří definovat jazyk $L \subseteq 0^*$ splňující následující dvě podmínky, bude důkaz u konce:

1) $L_k = L \Rightarrow \forall^* n \; s_k(n) \geq h(n-k)$

2) $\forall k\ \exists j\ L_j = L \wedge s_j(n) \leq h(n-k)$

Stačí potom totiž v první podmínce zvolit $k := i$ a v druhé $k := i+1$. Dostáváme tak $r(s_j(n)) \leq r(h(n-i-1)) = h(n-i) \overset{\forall^* n}{\leq} s_i(n)$.

Pro každé n vybereme nejvýš jeden prostorově nenáročný Turingův stroj a zajistíme, aby nerozpoznával jazyk L. Tento stroj $TM_{\sigma(n)}$ je zvolen tak, aby $\sigma(n)$ bylo nejmenší $i \leq n$ takové, že $s_i(n) < h(n-i)$ a $\forall i' < n\ \sigma(i') \neq i$. Aby Turingův stroj $TM_{\sigma(n)}$ nerozpoznával jazyk L, přidáme do L slovo 0^n, právě když $0^n \notin L_{\sigma(n)}$. Pokud $\sigma(n)$ není definováno, pak do L slovo 0^n nepřidáme. Řekneme, že $TM_{\sigma(n)}$ byl zrušen slovem délky n.

Existenční důkaz první podmínky: Existuje $n_0 > k$ takové, že $\forall j < k$ platí $\sigma(n) = j \Rightarrow n < n_0$. Potom však pro $n > n_0$ je $s_k(n) \geq h(n-k)$, nebo je TM_k zrušen slovem délky nejvýš n. Je-li však $L_k = L$, pak TM_k nemůže být nikdy zrušen.

Důkaz druhé podmínky: Rozmysleme si nejprve, co musí TM_j znát k rozhodnutí, zda slovo $0^n \in L$.

1 Pokud existuje $i = \sigma(n)$, je potřeba jej znát a zjistit, zda TM_i přijme 0^n nebo ne.

2 K zjištění, že $i = \sigma(n)$ potřebujeme zjistit, že $s_i(n) < h(n-i)$, že TM_i nebyl zrušen kratšími slovy a že žádné $i' < i$ nemá takovou vlastnost.

Algoritmus přijímající L může pracovat tak, že postupně pro každé m najde $\sigma(m)$ a poznamená si jej do seznamu zrušených strojů. Aby však algoritmus zjistil, zda $s_i(n) < h(n-i)$, potřebuje pro provedení simulace prostor $s_M(i,n) = n + (h(n-i) + \log n + 1)\lceil \log i \rceil$. TM_j ale smí použít prostor nejvýš $h(n-k)$. Protože $s_M(i,n) > h(n-i)$, TM_j nesmí simulovat stroje TM_i pro $i \leq k$.

Řešením je zvolit n_0 obdobně jako v důkazu první podmínky a konečně mnoho případů pro $n < n_0$ vyřešit v konečné jednotce. Číslo n_0 zvolíme tak, aby navíc platilo $2^{2^{n_0-k-2}} > n_0 > 64$ (tím zaručíme $h(n-k-1) > n$ a $(1+\log n)\lceil \log n \rceil < n$ pro $n \geq n_0$). Navíc je potřeba mít v konečné jednotce seznam všech zrušených strojů pro slova kratší než n_0. Algoritmus pak musí budovat seznam zrušených strojů až od $m > n_0$, nemusí přitom simulovat stroje TM_i pro $i \leq k$.

Pro prostor $s_M(i,m)$ potřebný na simulaci pro $n \geq m > n_0$ a $m \geq i > k$ platí $s_M(i,m) < n + (h(n-k-1) + \log n + 1)\lceil \log n \rceil < 2n + h(n-k-1)\lceil \log n \rceil < 2n + h(n-k-1)(n-1) < 2h(n-k-1) + h(n-k-1)\cdot(h(n-k-1)-2) = h(n-k-1)^2 \leq r(h(n-k-1)) = h(n-k)$.

Seznam zrušených strojů je dlouhý $n\lceil \log n \rceil < n^2 < h(n-k-1)^2 \leq r(h(n-k-1)) = h(n-k)$. Stroj TM_j vznikne z popsaného stroje použitím věty o lineární kompresi. □

7 Základní třídy složitosti (28. května 1998)

Definice 7.1

$$\boldsymbol{Log} = \boldsymbol{DSpace}(\log n)$$

$$\boldsymbol{NLog} = \boldsymbol{NSpace}(\log n)$$

$$\boldsymbol{PolyLog} = \bigcup_{i\geq 0} \boldsymbol{DSpace}(\log^i n)$$

$$\boldsymbol{P} = \bigcup_{i\geq 0} \boldsymbol{DTime}(n^i)$$

$$\boldsymbol{NP} = \bigcup_{i\geq 0} \boldsymbol{NTime}(n^i)$$

$$\boldsymbol{PSpace} = \bigcup_{i\geq 0} \boldsymbol{DSpace}(n^i)$$

$$\boldsymbol{NPSpace} = \bigcup_{i\geq 0} \boldsymbol{NSpace}(n^i)$$

$$\boldsymbol{DExT} = \bigcup_{c\geq 0} \boldsymbol{DTime}(2^{cn})$$

$$\boldsymbol{NExT} = \bigcup_{c\geq 0} \boldsymbol{NTime}(2^{cn})$$

$$\boldsymbol{ExpSpace} = \bigcup_{c\geq 0} \boldsymbol{DSpace}(2^{cn})$$

$$\boldsymbol{ExPTime} = \bigcup_{c\geq 0} \boldsymbol{DTime}(2^{n^c})$$

$$\boldsymbol{NExPTime} = \bigcup_{c\geq 0} \boldsymbol{NTime}(2^{n^c})$$

Poznámka 7.1 V názvu $\boldsymbol{DExT}$ se nevyskytuje písmeno $\boldsymbol{P}$, zatímco v názvu $\boldsymbol{ExPTime}$ ano. V prvním případě se v exponentu nevyskytuje polynom, zatímco v druhém ano.

Výskyt úvodního písmene $\boldsymbol{D}$ zdůrazňujícího determinismus je také nesystematické.

Poněkud divné je to s $\boldsymbol{ExpSpace}$, kde by člověk předpokládal polynom v exponentu.

Uvedené značení odpovídá značení z [1]. U větších tříd je ve značení nejednotnost. Proto když například někdo cizí mluví o exponenciálním čase, je vždy lepší se zeptat, zda je myslena třída $\boldsymbol{DTime}(2^{O(n)})$ nebo spíš $\boldsymbol{DTime}(2^{n^{O(1)}})$.

Toto byly definice tříd problémů (jazyků).

Pro deterministické Turingovy stroje je užitečné definovat třídy úloh (funkcí).

Definice 7.2 Úloha f patří do třídy úloh $\boldsymbol{DTimeF}(t)$, pokud existuje deterministický Turingův stroj, který pro libovolný vstup x spočítá $f(x)$ v čase t.

Úloha f patří do třídy úloh $\boldsymbol{DSpaceF}(s)$, pokud existuje deterministický Turingův stroj, který pro libovolný vstup x spočítá $f(x)$ s pracovním prostorem s.

Stejně tak jak odlišuje závěrečné $\boldsymbol{F}$ třídy úloh od tříd problémů (jazyků) v předcházejících definicích, budeme tuto konvenci používat i u odvozených (deterministických) tříd.

Definujme proto například

Definice 7.3

$$\boldsymbol{LogSpaceF} = \boldsymbol{DSpaceF}(\log n)$$

$$\boldsymbol{PF} = \bigcup_{i\geq 0} \boldsymbol{DTimeF}(n^i)$$

$$\boldsymbol{PSpaceF} = \bigcup_{i\geq 0} \boldsymbol{DSpaceF}(n^i)$$

8 Transformace, redukce a orákula (28. května 1998)

Úmluva 8.1 V této kapitole budeme psát o úlohách, i když se tato kapitola povětšině týká především problémů.

Při definici úplných (těžkých) úloh vůči nějaké třídě úloh jsme v úvodu do složitosti používali „polynomiálních transformací".

Definice „těžkosti" vůči nějaké třídě úloh byla motivována snahou nalézt takovou úlohu U, aby existence polynomiálního algoritmu pro úlohu U znamenala existenci polynomiálního algoritmu pro libovolnou úlohu z dané třídy. Z tohoto pohledu je nepřirozené definovat těžkost na základě transformací, protože transformace umožňují spustit výpočet úlohy U pouze jednou. Přirozenější je umožnit spouštět během polynomiálně dlouhého výpočtu výpočet úlohy U libovolně. Tomu se říká redukce.

Chceme–li definovat těžkost pomocí redukcí, je užitečné rozšířit si výpočetní model o používání tzv. orákula. Orákulum je nějaká daná funkce (úloha) O. Výpočetní model je rozšířen o pomocný prostor na vytváření parametrů pro orákulum a na čtení výsledků orákula. Kdykoli během výpočtu můžeme požádat orákulum, aby si přečetlo parametry a zapsalo výsledek funkce (úlohy) O.

Poznámka 8.1 Vzhledem k tomu, že orákulum může být libovolná funkce, můžeme při volbě vhodného orákula pracovat ve velmi silném výpočetním modelu. Toho se často využívá ve vyčíslitelnosti.

V těchto skriptech jsou orákula použita dvěma způsoby. Poprvé v kapitole věnované polynomiální hierarchii, později v druhé sekci kapitoly věnované interaktivním protokolům.

V prvním případě jsou orákula problémy, a místo pásky pro odpověď přechází stroj do stavu *O-ANO* případně *O-NE*. V druhém případě jsou orákula úlohy. Je vhodné si uvědomit, že oba dva modely jsou ekvivalentní máme-li k dispozici libovolný polynomiální čas. (Je možno použít sérii dotazů „Je odpověď úlohového orákula na otázku x aspoň i-bitová?", „Je i-tý bit odpovědi úlohového orákula na otázku x roven 1?".)

Těžkost můžeme pomocí redukcí (orákul) definovat následovně:

Definice 8.1 Úloha U je $\mathcal{T}$-těžká vůči polynomiální redukci, pokud je každá úloha z $\mathcal{T}$ řešitelná v polynomiálním čase s orákulem U.

Definice 8.2 Úloha U je $\mathcal{T}$-úplná vůči redukci, pokud je $U \in \mathcal{T}$ a U je $\mathcal{T}$-těžká vůči redukci.

Abychom rozlišili mezi těžkostí vůči redukci od těžkosti vůči transformaci, budeme v dalším značit $\mathcal{T}$-m-Poly-těžkost (úplnost), bude-li se jednat o těžkost (úplnost) vůči polynomiální transformaci. Těžkost (úplnost) vůči polynomiální redukci budeme značit $\mathcal{T}$-T-Poly-těžkost (úplnost).

Poznámka 8.2 Písmenko m vychází z 'many to one', písmenko T vychází z 'Turing reducibility'. Toto značení je přejato z terminologie vyčíslitelnosti. Ve vyčíslitelnosti je definována také 'one to one' převeditelnost, označovaná znakem 1, nicméně tu my potřebovat nebudeme.

Poznámka 8.3 Později definujeme převeditelnost transformací v logaritmickém meziprostoru. Těžké (úplné) úlohy vůči této převeditelnosti budeme značit $\mathcal{T}$-m-log-těžké (úplné).

Rozdíl mezi transformacemi a redukcemi je podstatný. Například T-stupně (třídy uzavřené na redukci) obsahují s každým jazykem i jeho doplňek. (Stačí znegovat odpověď orákula.)

Pokud budeme hovořit o výpočtech používajících orákulum, budeme používat zápisu $\boldsymbol{P}(O)$, k označení třídy jazyků rozpoznávaných v polynomiálním čase s orákulem O. Obdobně budeme značit $\boldsymbol{PSpace}(O)$ nebo $\boldsymbol{NP}(O)$.

9 Jiná interpretace průběhu nedeterministického výpočtu omezeného časem (6. července 1997)

V této kapitole obohatíme strukturu nedeterministického počítače tím, že rozdělíme stavy konečné jednotky do 4 významově odlišných skupin.

V první skupině budou tzv. *existenční* stavy, v druhé tzv. *všeobecné* stavy, ve třetí tzv. *náhodné* stavy. Čtvrtou skupinou jsou *deterministické* stavy, mezi něž patří všechny speciální koncové stavy *ANO*, *NE* a *NEVÍM*, ale také stavy *DOTAZ*, *O-ANO* a *O-NE*, sloužící na práci s orákuly.

Formálně $t : Q \to \{D, \exists, \forall, ?\}$ je funkce přiřazující stavům Turingova stroje jejich typ.

Definice 9.1 Jazyk L^A přijímaný obohaceným *TS* M, (takto obohacený *TS* budeme značit $\boxed{\exists\forall?}$-*TS*), definujeme následovně:

O výsledku výpočtu (zda je slovo x přijato, odmítnuto, nebo není známa odpověď) v čase $T(n)$ rozhoduje strom výpočtů M na vstupu x, omezený časem $T(n)$. ($T(n)$ je časově konstruovatelná.) S větví výpočtu, který neskončil v čase $T(n)$, zacházíme jako s výpočtem končícím ve stavu *NEVÍM*. (Dodáním počítadla do $T(n)$ můžeme M upravit tak, že M při dopočítání počítadla do stavu *NEVÍM* přejde.)

Výsledek výpočtu ν je definován pro každý uzel u stromu výpočtů na základě typu stavu s_u $\boxed{\exists\forall?}$-*TS* v uzlu u a na základě výsledků výpočtu v jednotlivých následnících $v_1, v_2, \ldots, v_k$ tohoto uzlu.

Výsledek výpočtu je trojice $\nu(u) = (a_u, r_u, q_u) \in \langle 0,1\rangle^3$, kde a_u reprezentuje pravděpodobnost přijetí, r_u pravděpodobnost zamítnutí a q_u pravděpodobnost, že algoritmus nedá odpověď. (Vždy platí $a_u + r_u + q_u = 1$.)

Výsledek výpočtu v uzlu reprezentujícím koncový stav je $(1,0,0)$ pro *ANO*, $(0,1,0)$ pro *NE* a $(0,0,1)$ pro *NEVÍM*.

Výsledek výpočtu v uzlu reprezentujícím existenční stav je

$$(t(s_u) = \exists) \Rightarrow \nu(u) = (\max a_{v_i}, \min r_{v_i}, 1 - a_u - r_u).$$

Výsledek výpočtu v uzlu reprezentujícím všeobecný stav je

$$(t(s_u) = \forall) \Rightarrow \nu(u) = (\min a_{v_i}, \max r_{v_i}, 1 - a_u - r_u).$$

Výsledek výpočtu v uzlu reprezentujícím náhodný stav je

$$(t(s_u) = ?) \Rightarrow \nu(u) = (\sum_{i=1}^{k} p_i a_{v_i}, \sum_{i=1}^{k} p_i r_{v_i}, \sum_{i=1}^{k} p_i q_{v_i}),$$

kde p_i je pravděpodobnost přechodu do následníka v_i.

Výsledek výpočtu v uzlu reprezentujícím deterministický stav je definován výsledkem výpočtu jediného následníka $\nu(u) = \nu(v_1)$, čemuž odpovídá chápání deterministického stavu jako stavu libovolného typu s jediným následníkem.

Na základě ν v kořeni ρ stromu výpočtu definujeme jazyky L^A a L^R:

$$x \in L^A \Leftrightarrow a_\rho > \frac{1}{2}, x \in L^R \Leftrightarrow r_\rho > \frac{1}{2}.$$

Pro jednoduchost se omezíme na takové $\boxed{\exists\forall?}$-*TS*, kde každý nedeterministický (rozuměj existenční, všeobecný nebo náhodný) stav má (nejvýš) dva následníky, a kde tito následníci jsou u náhodných stavů stejně pravděpodobní.

Poznámka 9.1 Uvědomme si vztah mezi *NTS* a $\boxed{\exists\forall?}$-*TS*. *NTS* můžeme chápat jako $\boxed{\exists}$-*TS*, tedy i jako $\boxed{\exists\forall?}$-*TS*, v němž jsou pouze existenční (a deterministické) stavy.

Definice 9.2 *Pravděpodobnost chyby* $e_M(x)$ $\boxed{\forall\exists?}$-*TS* M, je definována jako $r_\rho(x)$ pro $x \in L^A$ a jako $a_\rho(x)$ pro $x \in \in L^R$. Pokud $x \notin L^A \cup L^R$, pak $e_M(x) = 0$.

Pravděpodobnost neurčenosti $E_M(x)$ $\boxed{\forall\exists?}$-*TS* M, je definována jako $1 - a_\rho(x)$ pro $x \in L^A$ a jako $1 - r_\rho(x)$ pro $x \in L^R$. Pokud $x \notin L^A \cup L^R$, pak $E_M(x) = 1$.

Poznámka 9.2 Věta 9.1 ale ukazuje, že můžeme pracovat s takovými $\boxed{\forall\exists?}$-*TS*, pro něž je $E(x) = e(x)$. (Nutně pak každé x patří právě do jednoho z jazyků L^A, L^R.)

Definice 9.3 Typ je orientovaný graf, jehož vrcholy jsou ohodnoceny podmnožinami množiny $\{\forall, \exists, ?\}$. (Formálně typ je trojice (V, E, f), kde V je konečná množina, $E \subset \subset V \times V$ a $f : V \to \mathcal{P}(\{\forall, \exists, ?\})$.)

Turingův stroj M je strojem daného typu T (zkráceně M je T-*TS*), pokud existuje zobrazení $\tau : Q \to V$ zobrazující stavy Turingova stroje na příslušné vrcholy typu. (Formálně $t(q) \in f(\tau(q)) \cup \{D\}$.)

Navíc pokud přechodová funkce umožňuje přejít v jednom kroku ze stavu q do stavu q', pak stavům q, q' odpovídá tentýž vrchol typu, nebo je vrchol odpovídající stavu q spojen hranou s vrcholem odpovídajícím stavu q'. (Formálně $\tau(q) = \tau(q') \vee (\tau(q), \tau(q')) \in E$.)

Poznámka 9.3 Symbolem $\boxed{\ }$ značíme typ obsahující jediný vrchol, na nějž mohou být zobrazovány pouze deterministické stavy. $\boxed{\ }$-*TS* je totéž co *TS*.

Poznámka 9.4 Je-li typ T podtypem (příslušně ohodnoceným podgrafem) typu T', pak každý stroj typu T je strojem typu T'.

Poznámka 9.5 Dosud jsme se zmiňovali pouze o Turingových strojích typů, obsahujících jediný vrchol. V kapitole věnované polynomiální hierarchii nás budou zajímat Turingovy stroje typů, jejichž graf je cesta.

Definice 9.4 Dva typy jsou ekvivalentní, pokud pro libovolný stroj jednoho typu existuje stroj druhého typu přijímající tentýž jazyk v čase konstantakrát větším.

Cvičení 9.1 Nechť T je typ. Ukažte, že existuje ekvivalentní typ $T' = (V, E, f)$, kde (V, E) je acyklický graf s jediným zdrojem a s jediným stokem.

Ukažte, že libovolný stroj typu T' je možno za cenu zpomalení o konstantu simulovat strojem typu T', kde pro τ platí, že počáteční stav je zobrazen na zdroj typu T' a všechny koncové stavy jsou zobrazeny na stok typu T'.

Definice 9.5 Nechť T_1, T_2 jsou acyklické typy, každý s jediným zdrojem a stokem. Spojení typů (označíme symbolem $T_1 \to T_2$) vznikne sjednocením typů T_1 a T_2 a přidáním hrany ze stoku typu T_1 do zdroje typu T_2. Formálně, je-li s_1 stok typu $T_1 = (V_1, E_1, f_1)$ a z_2 zdroj typu $T_2 = (V_2, E_2, f_2)$, pak $T_1 \to T_2$ je trojice $(V_1 \cup V_2, E_1 \cup E_2 \cup \cup \{(s_1, z_2)\}, f)$, kde $f(v) = f_1(v)$ pro $v \in V_1$ a $f(v) = f_2(v)$ pro $v \in V_2$.

Poznámka 9.6 Uvědomme si, že spojení typů je asociativní. Symbol $\boxed{\exists} \to \boxed{\forall} \to \boxed{\exists}$ je typ.

Poznámka 9.7 Typy T a $\boxed{\ } \to T$ a $T \to \boxed{\ }$ jsou ekvivalentní. Pokud nás nezajímají typy, ale pouze jejich třídy ekvivalence, je $\boxed{\ }$ nulový prvek vůči skládání typů.

Věta 9.1 *Nechť t je časově konstruovatelná funkce a T typ. Je-li L jazyk přijímaný T-TS M v čase t, potom existuje takový T-TS M', přijímající jazyk L v čase $O(t)$, že pro libovolný vstup x platí $E_{M'}(x) = e_{M'}(x) < \frac{1}{2}$.*

Důkaz: Stroj M nejprve modifikujeme na M_1 tak, že stavy *NEVÍM* nahradíme stavy *NE*. (V čase t přejdeme též do stavu *NE* místo do *NEVÍM*.) Po této modifikaci je $q_\rho(x) = 0$, a tedy $a_\rho(x) + r_\rho(x) = 1$. Navíc se $a_\rho(x)$ nezměnilo. Pokud tedy bylo $x \in L$, je $x \in L^A_{M_1}$. Navíc pokud $x \in L^A_{M_1} \cup L^R_{M_1}$, pak $e(x) = E(x) < \frac{1}{2}$.

Jediný problém je, pokud $a_\rho = r_\rho = \frac{1}{2}$. Pak totiž $x \notin L^A_{M_1} \cup L^R_{M_1}$, $e(x) = 0$ a $E(x) = 1$. Neobsahuje-li M_1 náhodné stavy, tento problém nenastává. Zbývá tedy dokázat větu pro případ, kdy M_1 náhodné stavy obsahuje.

Přidáním časového počítadla zajistíme skončení libovolné větve výpočtu stroje M_1 v čase t. Pravděpodobnosti a_ρ i r_ρ můžeme zapsat ve tvaru $k/2^t$ pro vhodné k. Stroj M_1 přijímá, pokud $a_\rho > \frac{1}{2}$, M_1 nepřijímá, pokud $r_\rho \geq \frac{1}{2}$. Zajistíme-li, aby pro libovolný vstup x pro stroj M' platilo

$$a_\rho(x) = \frac{\frac{1}{2} + a_\rho(x)}{2} - \frac{1}{2^{t+2}}$$

a

$$r_\rho(x) = \frac{\frac{1}{2} + r_\rho(x)}{2} + \frac{1}{2^{t+2}},$$

bude M' přijímat tentýž jazyk jako M_1. Navíc $a_\rho \neq \frac{1}{2}$ a $r_\rho \neq \frac{1}{2}$.

Je-li $? \in f(\tau(q_0))$ (pro počáteční stav q_0), pak takové M' můžeme z M_1 vytvořit například takto: Místo časového počítadla do t použijeme počítadlo do $t+2$. Přidáme nový náhodný počáteční stav (místo původního počátečního stavu stroje M_1). Jedna větev přímo přechází v počáteční stav stroje M_1. Druhá větev výpočtu nezávisí na vstupu a platí pro ni $a = \frac{1}{2} - \frac{1}{2^{t+1}}$, $r = \frac{1}{2} + \frac{1}{2^{t+1}}$.

(To můžeme zajistit například tak, že jedna větev vede po rozskoku rovnou do stavu *NE* a druhá větev vede do stavu s, ve kterém buď náhodně zůstává nebo z něj pokračuje do stavu *ANO*. Po doběhnutí časového počítadla přechází M' ze stavu s do stavu *NE*.)

Pokud $? \notin f(\tau(q_0))$ (pro počáteční stav q_0), je konstrukce stroje M' nepatrně komplikovanější. Stroj M' si v konečné jednotce pamatuje bit informace, označující, zda byl již proveden rozskok. (Počet stavů se zdvojnásobí.) Pomocí tohoto bitu stroj M' zajistí, aby provedl rozskok, právě když se poprvé dostane do náhodného stavu. Jestliže pro x na vstupu bylo $a_\rho = r_\rho = \frac{1}{2}$, pak byly tyto hodnoty „zkopírovány“ (pomocí min a max) z nějakého náhodného stavu stromu výpočtu. Tento náhodný stav je první náhodný stav na nějaké cestě ve výpočetním stromu od jeho kořene. Proto je při výpočtu stroje M' proveden v tomto kroku rozskok, čímž je hodnota r zprůměrována s $\frac{1}{2} + \frac{1}{2^{t+1}}$, tedy větší než $\frac{1}{2}$. Pokud je při výpočtu stroje M' proveden rozskok a a_ρ bylo větší než $\frac{1}{2}$, pak zprůměrováním a_ρ s $\frac{1}{2} - \frac{1}{2^{t+1}}$ získáme opět číslo větší než $\frac{1}{2}$. Stroj M' tedy přijímá stejný jazyk jako stroj M_1 a každé slovo nepatřící do L^A patří do L^R. □

Definice 9.6 Nechť t je časově konstruovatelná funkce a T je typ. T-***Time***(t) je třída jazyků přijímaných T-*TS* v čase t.

$$T\text{-}\boldsymbol{P} = \bigcup_{i \geq 0} T\text{-}\boldsymbol{Time}(n^i)$$

Poznámka 9.8 Uvědomme si, že $\boxed{\ }$-***Time***(t) jsme dosud značili ***DTime***(t) a $\boxed{\ }$-$\boldsymbol{P}$ jsme značili $\boldsymbol{P}$. Obdobně $\boxed{\exists}$-***Time***(t) jsme dosud značili ***NTime***(t) a $\boxed{\exists}$-$\boldsymbol{P}$ jsme značili ***NP***.

Věta 9.2 *Libovolný $\boxed{\exists\forall?}$-TS pracující v čase t můžeme simulovat deterministicky v prostoru $O(t^2)$ a čase $O(t2^t)$.*

Důsledek 9.2.1

$$\boxed{\forall\exists?}\text{-}\boldsymbol{P} \subseteq \boldsymbol{PSpace}$$

Důkaz: Algoritmus Alg. 2 symbolicky popisuje simulaci. Dodejme ještě, že všechna čísla, se kterými pracujeme, jsou tvaru $\sum \frac{1}{2^i}$ pro $i < t$, což můžeme jednoduše reprezentovat řetězcem ne desetinných ale „polovinných“ míst těchto čísel. K tomu nám stačí řetězce délky t. Odtud velikost lokální paměti procedury je $O(t)$. Hloubka rekurze je t. Procedura je volána nejvýš $2 \cdot 2^t - 1$–krát.

Odtud dostáváme požadované odhady. □

---

procedure ***SimFEQTS***$(K : Konf) : ProbPair$;
begin
  TimeStep;
  if ***TimeOut*** **then return** $(0,0)$
  else
    if $s_u = det$ **then**
      case u **of**
        ANO : **return** $(1,0)$
        NE : **return** $(0,1)$
        $NEVÍM$: **return** $(0,0)$
        else : **return** ***SimFEQTS***$(\boldsymbol{NextDet}(K))$
      end
    else
      begin
        (a_0, r_0) := ***SimFEQTS***$(\boldsymbol{NextNedet}(0, K))$;
        (a_1, r_1) := ***SimFEQTS***$(\boldsymbol{NextNedet}(1, K))$;
        case s_u **of**
          $\exists$: **return** $(\max\{a_0, a_1\}, \min\{r_0, r_1\})$

$\forall$: **return** $(\min\{a_0, a_1\}, \max\{r_0, r_1\})$
$?$: **return** $\frac{1}{2}(a_0 + a_1, r_0 + r_1)$
end
end
end ;
begin

Vytvoř počáteční konfiguraci K, zaznamenej do ní i konfiguraci časového počítadla, spočítej

$$(a_\rho, r_\rho) := \boldsymbol{SimFEQTS}(K).$$

Je-li $a_\rho > \frac{1}{2}$ potom přijmi, je-li $r_\rho > \frac{1}{2}$, pak zamítni.
end

Alg. 2: Simulace $\boxed{\exists\forall?}$-*TS*

Poznámka 9.9 Předchozí věta by platila, i kdybychom omezili počet následníků nedeterministického vrcholu konstantou d a pokud by pravděpodobnosti následníků náhodných uzlů byly stejné.

Věta 9.3 *Libovolný* $\boxed{?}$*-TS pracující v čase* t *můžeme simulovat deterministicky v prostoru* $O(t)$ *a čase* $O(t2^t)$.

Důkaz: Vzhledem k omezení počtu následníků nedeterministického stavu na dva můžeme pro všechny možné řetězce délky t simulovat výpočet s tím, že v nedeterministických stavech se rozhodujeme podle následujícího znaku předepsaného řetězce. Navíc máme binární čítač pomocí něhož počítáme pravděpodobnosti přijetí a odmítnutí. Vždy, když řetězci odpovídá přijímací (odmítací) výpočet, zvětšíme přijímací (odmítací) počítadlo o 1 tak daleko zprava, kolik jsme ještě nepřečetli bitů řetězce. (Přičteme $2^{\text{počet nepřečtených bitů řetězce}} = 2^{t-\text{počet přečtených bitů}}$.) Hodnoty a_ρ (r_ρ) dostaneme po vydělení počítadel číslem 2^t. □

Poznámka 9.10 Pokud bychom omezili počet následníků náhodného vrcholu konstantou d a pokud by pravděpodobnosti následníků byly stejné, potom by předchozí věta platila s prostorem $O(t)$ a časem $O(td^t)$. Povolení pravděpodobností, které nejsou ve tvaru $\frac{1}{d}$ (pro malé přirozené d), nedává dobrý smysl.

Poznámka 9.11 Vzhledem k prostorovým nárokům algoritmu $O(t)$ je předem jasný odhad času $2^{O(t)}$. Přesnějším (amortizovaným) rozborem práce binárního počítadla bychom dostali časový odhad $O(2^t)$.

Poznámka 9.12 Simulace pomocí stejného typu *TS* je možná v čase $O(t \log s)$ a prostoru $O(s)$, jak bylo ukázáno v kapitole o simulacích.

Poznámka 9.13 Připomeňme, že věty o universálních Turingových strojích jsme vyslovili pro libovolný typ T.

Simulace je závislá na typu pouze v tom, pomocí jakého typu nedeterministického stavu vybíráme následující „rozšířený display“.

Aby byl výsledek správně interpretován, je následník stavu q vybírán stavem typu $t(q)$ universálního stroje. Vždy je použit stav q_u universálního stroje, pro nějž je $\tau(q_u) = \tau(q)$. K tomu stačí pro každý vrchol v typu T jeden nedeterministický stav pro každý typ stavu z $f(v)$.

Když funkce τ není známa, musíme nejprve nějaké τ nalézt. Nároky na nalezení τ nejsou podstatné, protože jsou konstantní vůči velikosti vstupu simulovaného stroje. (Uhodnutí τ a ověření, že odpovídá přechodové funkci, stojí čas $|T|^{|Q|} \cdot |\delta|$.)

10 Třídy definované pomocí $\boxed{?}$-*TS* pracujících v polynomiálním čase (29. dubna 1998)

Definice 10.1 Třída ***PP*** obsahuje jazyky přijímané $\boxed{?}$-*TS* v polynomiálním čase.

Poznámka 10.1 V minulé kapitole jsme třídu ***PP*** nazvali $\boxed{?}$-***P***.

Poznámka 10.2 Podle věty 9.1 se v definici třídy ***PP*** můžeme omezit na $\boxed{?}$-*TS*, kde $e(x) = E(x) < \frac{1}{2}$.

Věta 10.1 *Platí* $\boldsymbol{NP \subseteq PP \subseteq PSpace}$.

Důkaz: Nechť $L \in \boldsymbol{NP}$. Z $\boxed{\exists}$-*TS* M přijímajícího L vytvoříme následující $\boxed{?}$-*TS*. Změníme typ každého $\exists$ stavu na $?$ stav. Tím dostaneme stroj $\boxed{?}$-*TS* M'. Poté přidáme počáteční náhodný stav, z něhož přecházíme s pravděpodobností $\frac{1}{2}$ do stavu *ANO* a se stejnou pravděpodobností do počátečního stavu stroje M'. Tak vznikne $\boxed{?}$-*TS* M''.

Platí $x \in L$, právě když $a_\rho(x, M') > 0$, a tedy právě když $a_\rho(x, M'') > \frac{1}{2}$, a tedy právě když slovo x je přijímáno $\boxed{?}$-*TS* M''.

Druhá inkluze je důsledkem věty 9.3. □

Věta 10.2 ***PP*** *je uzavřena na komplementy.*

Důkaz: Použijeme $\boxed{?}$-*TS* M, pro nějž $e_M(x) = E_M(x) < \frac{1}{2}$. Stačí potom pouze zaměnit stavy *ANO* za *NE*. □

Důsledek 10.2.1 *Platí* $\boldsymbol{NP \cup co\text{-}NP \subseteq PP}$.

Není známo, zda je ***PP*** uzavřena na průniky a sjednocení. Z neuzavřenosti by ihned plynulo $\boldsymbol{NP \neq PP \neq PSpace}$.

Věta 10.3 ***PP*** *je uzavřena na symetrický rozdíl.*

Důkaz: Nechť $A, B \in \boldsymbol{PP}$, nechť A je přijímáno $\boxed{?}$-*TS* M_A, nechť B je přijímáno $\boxed{?}$-*TS* M_B (oboje v polynomiálním čase). Nechť $e_{M_A}(x) = E_{M_A}(x) < \frac{1}{2}$, $e_{M_B}(x) = E_{M_B}(x) < \frac{1}{2}$.

Zkonstuujeme $\boxed{?}$-*TS* M, který nejprve provede M_A, poté M_B a přijímá, právě když právě jeden z výpočtů přijímal. Ověříme, že $\boxed{?}$-*TS* M přijímá $A \triangle B$.

Odhadněme $e(x)$ — pravděpodobnost, že x je přijato chybně: slovo x je $\boxed{?}$-*TS* M přijímáno chybně, pokud je x přijímáno chybně právě jedním ze $\boxed{?}$-*TS* M_A, M_B.

Odtud

$$e(x) = e_{M_A}(x) \cdot (1 - e_{M_B}(x)) + (1 - e_{M_A}(x)) \cdot e_{M_B}(x).$$

Vzhledem k tomu, že funkce $x(1-y) + (1-x)y$ zobrazuje interval $\langle 0, \frac{1}{2})^2$ do intervalu $\langle 0, \frac{1}{2})$, dostáváme $e(x) < \frac{1}{2}$. Vzhledem k tomu, že $e(x) < \frac{1}{2}$, je $L_M^A = A \triangle B$ a $e(x) = e_M(x) = E_M(x)$. □

Definice 10.2 Třída ***BPP*** obsahuje jazyky přijímané v polynomiálním čase $\boxed{?}$-*TS*, kde

$$e(x) = E(x) < \varepsilon < \frac{1}{2}$$

Věta 10.4 *Platí* $\boldsymbol{BPP \subseteq PP}$. □

Definice 10.3 Třída ***R*** (jindy označovaná také ***RP*** nebo ***VPP***), obsahuje jazyky L přijímané $\boxed{?}$-*TS* v polynomiálním čase s nulovou pravděpodobností neurčenosti pro slova nepatřící do jazyka.

$$x \notin L \Rightarrow E(x) = 0$$

Poznámka 10.3 Pokud $\boxed{?}$-*TS* M rozpoznávající jazyk L „ve smyslu“ ***R*** skončil pro vstup x ve stavu *ANO*, potom $x \in L$.

Věta 10.5 *Nechť p je polynom. Polynomiálním iterováním algoritmu můžeme pravděpodobnost chyby pro jazyk A z* ***BPP*** *(resp.* ***R****) snížit na* $(\frac{1}{2})^{p(|x|)}$.

Důkaz: Iterovat jazyk z ***R*** je jednoduché, spustíme výpočet $p(|x|)-$krát a přijmeme, pokud některý výpočet přijme. (Je-li $\varepsilon^c < \frac{1}{2}$, pak pomocí c iterací snížíme pravděpodobnost chyby pod $\varepsilon' < \frac{1}{2}$, proto $\frac{1}{2}$ není v definici přijímání $\boxed{?}$-*TS* pro třídu ***R*** podstatná.)

Iterovat jazyk z ***BPP*** je těžší: Nechť $q(n) = c \cdot p(n)$, kde $(4\varepsilon(1-\varepsilon))^c < \frac{1}{2}$. Spustíme výpočet $(2q(n)+1)-$ $-$krát a slovo přijmeme, pokud nadpoloviční počet výpočtů slovo přijal. (Spustíme výpočet $k-$krát je třeba interpretovat tak, že přidáme počítadlo od k a vždy místo přechodu do koncového stavu zaznamenáme druh koncového stavu, snížíme počítadlo a je-li nenulové vrátíme se do „počáteční konfigurace“. Do koncového stavu přejdeme na základě vyhodnocení zaznamenaných koncových stavů.)

Pravděpodobnost chyby je

$$\sum_{j=0}^{q} \binom{2q+1}{j} \cdot (1-\varepsilon)^j \cdot \varepsilon^{2q+1-j} < (1-\varepsilon)^q \cdot \varepsilon^{q+1} \cdot \sum_{j=0}^{q} \binom{2q+1}{j} <$$

$$< (1-\varepsilon)^q \cdot \varepsilon^{q+1} \cdot 2^{2q+1} = 2\varepsilon((1-\varepsilon)\cdot\varepsilon\cdot 4)^{c\cdot p} < 2\varepsilon\left(\frac{1}{2}\right)^p < \left(\frac{1}{2}\right)^p$$

□

Věta 10.6 *Platí* $\boldsymbol{P \subseteq R \subseteq NP \cap BPP}$.

Důkaz: Jediná netriviální inkluze je $\boldsymbol{R \subseteq NP}$. Je-li $L \in \boldsymbol{R}$ a je-li M $\boxed{?}$-*TS*, který toto zaručuje, pak $\boxed{\exists}$-*TS* M zaručuje $L \in \boldsymbol{NP}$. □

Věta 10.7 *Třída* ***BPP*** *je uzavřena na doplňky, sjednocení a průniky, třída* ***R*** *je uzavřena na sjednocení a průniky.*

Důkaz: Uzavřenost ***BPP*** na doplňky dokážeme přehozením významu stavů *ANO* a *NE* po úpravě stroje do tvaru, kde $e(x) = E(x) < \frac{1}{2}$.

Jediným trikem v druhé části důkazu je pro dané $\varepsilon < \frac{1}{2}$ zvolit $\boxed{?}$-*TS* pracující s chybou $\frac{\varepsilon}{2}$. Algoritmus potom pustí simulaci obou algoritmů a zařídí se podle výsledku. Pravděpodobnost chyby je nejvýš součet pravděpodobností chyb. □

Definice 10.4 ***co-R*** je třída jazyků, jejichž doplňky jsou v $\boldsymbol{R}$.

Poznámka 10.4 Pro následující definici je vhodný pojem *jazyk rozpoznávaný* Turingovým strojem — L je jazyk rozpoznávaný Turingovým strojem M, právě když $L = L_M^A$ a pro libovolný vstup x platí $x \in L_M^A \cup L_M^R$.

Definice 10.5 ***ZPP*** je třída jazyků L, rozpoznávaných $\boxed{?}$-*TS* v polynomiálním čase a s nulovou chybou.

$$\big(x \in L \Rightarrow (a_\rho > \frac{1}{2} \wedge r_\rho = 0)\big) \wedge \big(x \notin L \Rightarrow (a_\rho = 0 \wedge r_\rho > \frac{1}{2})\big)$$

Věta 10.8 *Třída* ***ZPP*** *je uzavřena na doplňky, sjednocení i průniky.*

Důkaz: Iterováním můžeme pro libovolné $\varepsilon > 0$ zajistit $q_\rho < \varepsilon$. Přehozením stavů *ANO* a *NE* potom dostaneme požadovaný $\boxed{?}$-*TS*. Uzavřenost na sjednocení a průniky stejně jako pro třídy ***BPP*** a ***R*** dostaneme tak, že spustíme postupně oba $\boxed{?}$-*TS* a na základě jejich výsledků určíme výsledek operace. □

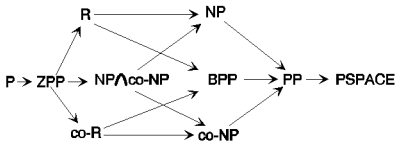
Věta 10.9 *Platí* $\boldsymbol{ZPP = R \cap co\text{-}R}$.

Důkaz: Stav *NEVÍM* můžeme nahradit stavem *NE*, tím změníme stroj přijímající ve smyslu ***ZPP*** na stroj přijímající ve smyslu ***R***. Odtud $\boldsymbol{ZPP \subseteq R}$. Z uzavřenosti ***ZPP*** na doplněk plyne zbývající část první inkluze.

Nechť naopak $L \in \boldsymbol{R} \cap \boldsymbol{co\text{-}R}$. Jinými slovy L i $\overline{L}$ jsou v ***R***. Označme M_L (resp. $M_{\overline{L}}$) $\boxed{?}$-*TS*, podle nichž je L (resp. $\overline{L}$) v ***R***. Zkonstruujeme $\boxed{?}$-*TS* M, který ukáže, že $L \in \boldsymbol{ZPP}$. M nejprve simuluje M_L, potom $M_{\overline{L}}$. Pokud M_L přijme, M přejde do stavu *ANO*, pokud $M_{\overline{L}}$ přijme, M přejde do stavu *NE*. V ostatních případech M přejde do stavu *NEVÍM*. □

Důsledek 10.9.1 *Platí* $\boldsymbol{P \subseteq ZPP \subseteq NP \cap co\text{-}NP}$.

Důkaz: Protože $\boldsymbol{P \subseteq R \subseteq NP}$, je $\boldsymbol{P \subseteq R \cap co\text{-}R \subseteq NP \cap}$ $\boldsymbol{\cap\ co\text{-}NP}$. □



Obr. 2: Vztahy mezi pravděpodobnostními třídami.

Vztahy mezi jednotlivými třídami zachycuje obrázek 2.

11 Polynomiální hierarchie (29. září 1997)

Připomeňme značení $\boldsymbol{P}(A)$, $\boxed{\exists}$-$\boldsymbol{P}(A)$ a $\boxed{\forall}$-$\boldsymbol{P}(A)$ jazyků rozpoznávaných v polynomiálním čase po řadě deterministickým turingovým strojem, $\boxed{\exists}$-TS a $\boxed{\forall}$-TS vždy s orákulem A. (Alternativní značení je $\boldsymbol{NP}(A)$ pro $\boxed{\exists}$-$\boldsymbol{P}(A)$ a $\boldsymbol{co}$-$\boldsymbol{NP}(A)$ pro $\boxed{\forall}$-$\boldsymbol{P}(A)$.)

Poznámka 11.1 Platí $L \in \boxed{\exists}$-$\boldsymbol{P}(A)$, právě když $\overline{L} \in \boxed{\forall}$-$\boldsymbol{P}(A)$. (Stačí odstranit stavy *NEVÍM* a zaměnit stav *ANO* za *NE* a typy stavů existenční za všeobecné.)

Zaveďme obdobné značení pro třídu problémů $\mathcal{T}$. Nechť $\boldsymbol{P}(\mathcal{T})$, $\boxed{\exists}$-$\boldsymbol{P}(\mathcal{T})$ a $\boxed{\forall}$-$\boldsymbol{P}(\mathcal{T})$ označuje jazyky rozpoznávané v polynomiálním čase příslušnými stroji s vhodně zvoleným orákulem $A \in \mathcal{T}$.

Definice 11.1 Nechť $\mathcal{T}$ je třída jazyků. Jazyk L je $\mathcal{T}$-T-poly-těžký, právě když $\mathcal{T} \subseteq \boldsymbol{P}(L)$. Jazyk L je $\mathcal{T}$-T-poly-úplný, právě když zároveň L je $\mathcal{T}$-T-poly-těžký a $L \in \mathcal{T}$.

Poznámka 11.2 Připomeňme že L je $\mathcal{T}$-m-poly-těžký, právě když pro libovolný jazyk $L' \in \mathcal{T}$ existuje transformace $f \in \boldsymbol{PF}$, taková, že $x \in L' \Leftrightarrow f(x) \in L$.

Každý $\mathcal{T}$-m-poly-těžký jazyk je $\mathcal{T}$-T-poly-těžký (ale nejspíš ne naopak). (Zapíšeme na pásku orákula $f(x)$ a přejdeme do stavu *DOTAZ*. Ze stavu *O-ANO* přejdeme ihned do stavu *ANO* a ze stavu *O-NE* ihned do stavu *NE*.)

Poznámka 11.3 Je-li S $\mathcal{T}$-T-úplný problém, pak $\boldsymbol{P}(\mathcal{T}) = \boldsymbol{P}(S)$, $\boxed{\exists}$-$\boldsymbol{P}(\mathcal{T}) = \boxed{\exists}$-$\boldsymbol{P}(S)$ a $\boxed{\forall}$-$\boldsymbol{P}(\mathcal{T}) = \boxed{\forall}$-$\boldsymbol{P}(S)$.

(Nechť T je typ. Je-li $L \in T$-$\boldsymbol{P}(\mathcal{T})$, pak pro nějaké $O \in \mathcal{T}$ je $L \in T$-$\boldsymbol{P}(O)$. Protože $O \in \boldsymbol{P}(S)$, můžeme polynomiálně mnoho dotazů typu „$x \in O$?“ odpovědět s orákulem S v celkově polynomiálním čase.)

Poznámka 11.4 $\emptyset$, neboli problém, jehož cílem je na libovolný dotaz odpovědět ne, je $\boldsymbol{P}$-T-úplný problém.

Definice 11.2 Definujme následovně hierarchii tříd:

1. $\Sigma_0^P = \Pi_0^P = \Delta_0^P = \boldsymbol{P} = \boldsymbol{P}(\emptyset)$;
2. $\Sigma_{k+1}^P = \boxed{\exists}$-$\boldsymbol{P}(\Sigma_k^P)$ pro $k \geq 0$;
3. $\Delta_{k+1}^P = \boldsymbol{P}(\Sigma_k^P)$ pro $k \geq 0$;
2. $\Pi_{k+1}^P = \boxed{\forall}$-$\boldsymbol{P}(\Sigma_k^P)$ pro $k \geq 0$;

Definujme také

$$\boldsymbol{PH} = \bigcup_{k \geq 0} \Sigma_k^P = \bigcup_{k \geq 0} \Pi_k^P = \bigcup_{k \geq 0} \Delta_k^P.$$

Třída $\boldsymbol{PH}$ se nazývá *polynomiální hierarchie*.

Poznámka 11.5 Horní index ^{P} odlišuje tuto hierarchii od aritmetické, případně Booleovské hierarchie. V dalším tento index budeme vynechávat, protože o těchto hierarchiích již nebude v těchto skriptech zmínka.

V této kapitole zevedeme tři další definice tříd polynomiální hierarchie. Než dokážeme ekvivalenci takto definovaných tříd, budeme je rozlišovat indexy $^{(1)}$, $^{(2)}$, $^{(3)}$, $^{(4)}$, které budeme psát místo indexu ^{P}. Pro již uvedenou definici budeme používat index $^{(1)}$.

Poznámka 11.6 Polynomiální hierearchii můžeme také definovat relativně vůči nějakému orákulu A. Odlišnost v definici je v tom, že za „nultou“ třídu bereme místo $\boldsymbol{P}(\emptyset)$ třídu $\boldsymbol{P}(A)$.

Nás relativizované verze polynomiální hierarchie zajímat nebudou.

Lemma 11.1 *Platí*

(a) $\Delta_1^{(1)} = \boldsymbol{P}$

(b) $\Pi_k^{(1)} = \boldsymbol{co}$-$\Sigma_k^{(1)}$

(c) $\Sigma_{k+1}^{(1)} = \boxed{\exists}$-$\boldsymbol{P}(\Pi_k^{(1)})$

(d) $\Delta_{k+1}^{(1)} = \boldsymbol{P}(\Pi_k^{(1)})$

(e) $\Pi_{k+1}^{(1)} = \boxed{\forall}$-$\boldsymbol{P}(\Pi_k^{(1)})$

(f) $\Sigma_{k+1}^{(1)} = \boxed{\exists}$-$\boldsymbol{P}(\Delta_{k+1}^{(1)})$

(g) $\Pi_{k+1}^{(1)} = \boxed{\forall}$-$\boldsymbol{P}(\Delta_{k+1}^{(1)})$

Důkaz:

(a) $\Delta_1^{(1)} = \boldsymbol{P}(\boldsymbol{P}) = \boldsymbol{P}(\emptyset) = \boldsymbol{P}$. (Využili jsme poznámek 11.3 a 11.4.)

(b) Přímo z definice, s využitím poznámky 11.1

(c),(d),(e) S použitím (b), stačí zaměnit stavy *O-ANO* za *O-NE*, do nichž přechází stroj po odpovědích orákula.

(f),(g) Je-li $B \in \Delta_{k+1}^{(1)} = \boldsymbol{P}(\Sigma_k^{(1)})$, pak odpovědi orákula B můžeme odsimulovat deterministicky v polynomiálním čase pomocí orákula z $\Sigma_k^{(1)}$. □

Lemma 11.2 *Platí* $\Sigma_k^{(1)} \cup \Pi_k^{(1)} \subseteq \Delta_{k+1}^{(1)} \subseteq \Sigma_{k+1}^{(1)} \cap \Pi_{k+1}^{(1)}$.

Důkaz: $\Delta_{k+1}^{(1)} = \boldsymbol{P}(\Sigma_k^{(1)}) = \boldsymbol{P}(\Pi_k^{(1)})$, odtud plyne první inkluze.

Druhá inkluze plyne z toho, že $\boldsymbol{P}(A) = \boxed{}$-$\boldsymbol{P}(A) \subseteq \boxed{\exists}$-$\boldsymbol{P}(A)$, $\boldsymbol{P}(A) = \boxed{}$-$\boldsymbol{P}(A) \subseteq \boxed{\forall}$-$\boldsymbol{P}(A)$ pro libovolné orákulum A. □

Pomocí kvantifikovaných predikátů můžeme polynomiální hierarchii vyjádřit následujícím způsobem:

Definice 11.3

$$L \in \Sigma_k^{(2)} \Leftrightarrow L = \{x \mid \exists^p y_1 \forall^p y_2 \exists^p y_3 \cdots y_k \; R^p(x, y_1, y_2, \cdots, y_k)\},$$

$$L \in \Pi_k^{(2)} \Leftrightarrow L = \{x \mid \forall^p y_1 \exists^p y_2 \forall^p y_3 \cdots y_k \; R^p(x, y_1, y_2, \cdots, y_k)\},$$

kde pro daný polynom p je R^p nějaký predikát z $\boldsymbol{DTime}(p)$, „$\exists^p y$“ je zkratkou pro „$\exists\, y\ |y| < p(|x|) \wedge$“ a „$\forall^p y$ je zkratkou pro $\forall\, y\ |y| < p(|x|) \Rightarrow$“ (výraz je uzávorkován zprava).

Definice 11.4 Označme σ_k typ $\boxed{\exists} \to \boxed{\forall} \to \boxed{\exists} \to \cdots$ obsahující právě k vrcholů. Označme π_k typ $\boxed{\forall} \to \boxed{\exists} \to \to \boxed{\forall} \to \cdots$ oksahující právě k vrcholů.

$$\Sigma_k^{(3)} = \sigma_k\text{-}\boldsymbol{P}$$

$$\Pi_k^{(3)} = \pi_k\text{-}\boldsymbol{P}.$$

Definice 11.5 Nechť $\mathcal{C}$ je třída jazyků. Pak $\exists\mathcal{C}$ (resp. $\forall\mathcal{C}$) je třída jazyků obsahující jazyk A, právě když existuje jazyk $B \in C$ a polynom p tak, že $x \in A$, právě když $\exists\, y\ |y| < p(|x|) \wedge \langle x, y\rangle \in B$ (resp. $\forall\, y\ |y| < p(|x|) \Rightarrow \langle x, y\rangle \in B$).

$$\Sigma_k^{(4)} = \exists\forall\exists\cdots\boldsymbol{P}$$

$$\Pi_k^{(4)} = \forall\exists\forall\cdots\boldsymbol{P},$$

kde je v obou případech použito právě k kvantifikátorů.

Definice 11.6 Nechť L je jazyk. Definujme jazyk $L^{(*)}$ následovně:

$$L^{(*)} = \left\{x \mid \exists\, i\ x = \langle x_1, x_2, \ldots, x_i\rangle \wedge \forall\, j\ x_j \in L\right\}$$

Lemma 11.3 *Platí* $L \in \boxed{\exists}\text{-}\boldsymbol{P}(O) \Leftrightarrow L^{(*)} \in \boxed{\exists}\text{-}\boldsymbol{P}(O)$ a $L \in \boxed{\forall}\text{-}\boldsymbol{P}(O) \Leftrightarrow L^{(*)} \in \boxed{\forall}\text{-}\boldsymbol{P}(O)$

Důkaz: Máme-li stroj přijímající $L^{(*)}$, dotazem na $\langle x\rangle$ zjistíme zda x patří do L. Opačně, máme-li T-TS(O) M přijímající L v čase p, zkonstruujeme stroj M', který pro vstup $\langle x_1, \ldots, x_i\rangle$ pouští pro každé i stroj M. V případě, že pro některé j výpočet neskončí kladnou odpovědí v čase p, výsledkem je *NE*. Pokud všechny výpočty skončí kladně, výsledek je *ANO*. □

Lemma 11.4 *Platí* $\Sigma_k^{(1)} = \Sigma_k^{(4)}$ a $\Pi_k^{(1)} = \Pi_k^{(4)}$.

Důkaz: Důkaz provedeme indukcí podle k. Pro $k = 0$ se všechny uvedené třídy rovnají $\boldsymbol{P}$, takže tvrzení platí. Nechť $k \geq 0$. Tvrzení dokážeme pro Σ_{k+1}. Pro Π_{k+1} je důkaz analogický.

Nechť $L \in \Sigma_{k+1}^{(1)}$. Potom existuje polynom p, orákulum $O \in \Pi_k^{(1)}$ a $\boxed{\exists}$-$TS(O)$ M, tak, že M přijímá L v čase p. Pro dané x sestavíme predikát, který bude pravdivý, právě když M přijme slovo x. Stroj M přijme slovo x, pokud existuje přijímací větev výpočtu (označíme y řetězec všech nedeterministických rozhodnutí). Ke kontrole toho, že y vede k přijetí slova x, je potřeba znát odpovědi orákula na dotazy položené v průběhu výpočtu. Nechť $p_1, \ldots, p_{i_p}$ (resp. $n_1, \ldots, n_{i_n}$) jsou všechny kladně (resp. záporně) zodpovězené dotazy v průběhu větve výpočtu y. Nechť $R(x, y, \langle p_1, \ldots, p_{i_p}\rangle, \langle n_1, \ldots, n_{i_n}\rangle)$ je predikát, který zkontroluje, že y je přijímací větví výpočtu za předpokladu, že $p_1, \ldots, p_{i_p}$ (resp. $n_1, \ldots, n_{i_n}$) jsou všechny kladně (resp. záporně) zodpovězené dotazy v průběhu větve výpočtu y. Ke kontrole, zda $p_1, \ldots, p_{i_p}$ jsou orákulem O odpovězena kladně, stačí otestovat $\langle p_1, \ldots, p_{i_p}\rangle \in O^{(*)}$. Ke kontrole, zda $n_1, \ldots, n_{i_n}$ jsou orákulem O odpovězena záporně, stačí otestovat $\langle n_1, \ldots, n_{i_n}\rangle \notin O^{(*)}$. V případě $k = 0$ jsou oba tyto dotazy predikáty z $\boldsymbol{P}$. Pro $k > 0$ je vzhledem k indukčnímu předpokladu $\Sigma_k^{(1)} = \Sigma_k^{(4)} = \exists\Pi_{k-1}^{(4)}$, a tedy pro nějaký jazyk $O' \in \Pi_{k-1}^{(4)}$ je $\langle n_1, \ldots, n_{i_n}\rangle \notin$ $\notin O^{(*)} \Leftrightarrow \exists t\ \langle\langle n_1, \ldots, n_{i_n}\rangle, t\rangle \in O'$. Predikát R' definovaný vztahem $\langle x, \langle y, \langle p_1, \ldots, p_{i_p}\rangle, \langle n_1, \ldots, n_{i_n}\rangle, t\rangle\rangle \in$ $\in R' \Leftrightarrow R\big(x, y, \langle p_1, \ldots, p_{i_p}\rangle, \langle n_1, \ldots, n_{i_n}\rangle\big) \wedge \langle p_1, \ldots, p_{i_p}\rangle \in$ $\in O^{(*)} \wedge \langle\langle n_1, \ldots, n_{i_n}\rangle, t\rangle \in O'$ je průnik tří predikátů z $\Pi_k^{(4)} \overset{\text{IP}}{=} \Pi_k^{(1)}$. Vzhledem k uzavřenosti $\boxed{\forall}$-$\boldsymbol{P}(O)$ na průnik je tento predikát v $\Pi_k^{(1)}$. Platí, že x je přijato strojem M právě když $\exists\, w\ \langle x, w\rangle \in R'$. ($R'$ přijme $\langle x, w\rangle$ pouze tehdy, je-li w tvaru $\langle y, \langle p_1, \ldots, p_{i_p}\rangle, \langle n_1, \ldots, n_{i_n}\rangle, t\rangle$) Tedy $L \in$ $\in \exists\Pi_k^{(1)} \overset{\text{IP}}{=} \exists\Pi_k^{(4)} = \Sigma_{k+1}^{(4)}$.

Nechť $L \in \Sigma_{k+1}^{(4)} = \exists\Pi_k^{(4)} \overset{\text{IP}}{=} \exists\Pi_k^{(1)}$. Existuje tedy polynom p a jazyk $L' \in \Pi_k^{(1)}$ tak, že $x \in L \Leftrightarrow \exists\, y\ |y| < p(|x|) \wedge \langle x, y\rangle \in$ $\in L'$. Zkonstruujeme $\boxed{\exists}$-$TS(L')$ M přijímající L v čase $O(p)$. Stroj M nedeterministicky generuje bity y, případně konec slova y. Přidáním „hodin do $p(|x|)$“ je omezena délka řetězce y. Nakonec M vydá odpověď shodně s odpovědí orákula na dotaz $\langle x, y\rangle$. Odtud $L \in \exists$-$\boldsymbol{P}(L') \subseteq \exists$-$\boldsymbol{P}(\Pi_k^1) =$ $= \Sigma_{k+1}^{(1)}$. □

Lemma 11.5 *Platí* $\Sigma_k^{(2)} = \Sigma_k^{(4)}$ a $\Pi_k^{(2)} = \Pi_k^{(4)}$.

Důkaz: Indukcí: Provedeme pouze pro Σ, pro Π je důkaz obdobný. Pro $k = 0$ tvrzení platí.

Nechť $L \in \Sigma_{k+1}^{(4)} = \exists\Pi_k^{(4)} \overset{\text{IP}}{=} \exists\Pi_k^{(2)}$. Existuje tedy polynom p a jazyk $L' \in \Pi_k^{(2)}$ tak, že $x \in L \Leftrightarrow \exists\, y\ |y| <$ $< p(|x|) \wedge \langle x, y\rangle \in L'$. Pro L' existuje vyjádření tvaru $x \in L' \Leftrightarrow \forall\, y_1\ \exists\, y_2\ \cdots\ y_k\ R(x, y_1, \cdots, y_k)$. Tedy $x \in$ $\in L \Leftrightarrow \exists\, y\ \forall\, y_1\ \exists\, y_2\ \cdots\ y_k\ R(\langle x, y\rangle, y_1, \cdots, y_k) =$ $= \exists\, y_1\ \forall\, y_2\ \cdots\ y_{k+1}\ R'(x, y_1, \cdots, y_{k+1})$. Predikát R' transformujeme v polynomiálním čase na predikát R „odstraněním závorek“. Tedy $L \in \Sigma_{k+1}^{(2)}$.

Nechť naopak $L \in \Sigma_{k+1}^{(2)}$. Pro L tedy existuje vyjádření ve tvaru $x \in L \Leftrightarrow \exists^p\, y_1\ \forall^p\, y_2\ \cdots\ y_{k+1}\ R^p(x, y_1, \ldots, y_{k+1})$. Definujme jazyk L' následovně: $\langle x, y_1\rangle \in L' \Leftrightarrow$ $\Leftrightarrow \forall^p\, y_2\ \cdots\ y_{k+1}\ R^p(x, y_1, \ldots, y_{k+1})$. Evidentně je $L' \in$ $\in \Pi_k^{(2)}$. Ale $x \in L \Leftrightarrow \exists^p\, y_1\ \langle x, y_1\rangle \in L'$, tedy $L \in$ $\in \exists\,\Pi_k^{(2)} \overset{\text{IP}}{=} \exists\,\Pi_k^{(4)} = \Sigma_{k+1}^{(4)}$. □

Lemma 11.6 *Pro libovolný typ T platí*

$$\exists\,(T\text{-}\boldsymbol{P}) = (\boxed{\exists} \to T)\text{-}\boldsymbol{P} \text{ a } \forall\,(T\text{-}\boldsymbol{P}) = (\boxed{\forall} \to T)\text{-}\boldsymbol{P}.$$

Důkaz: Nechť $L \in \exists\,(T$-$\boldsymbol{P})$, nebo-li existuje polynom p a $L' \in T$-$\boldsymbol{P}$, tak, že $x \in L \Leftrightarrow \exists\, y\ |y| < p(|x|) \wedge \langle x, y\rangle \in L'$. Vzhledem k tomu, že $L' \in T$-$\boldsymbol{P}$, existuje polynom p' a T-TS M' přijímající L' v čase p'. Vytvoříme $(\boxed{\exists} \to T)$-TS M přijímající jazyk L v čase $p+p'$. Stroj M nejprve vygeneruje y délky nejvýše $p(|x|)$ pomocí existenčních stavů (a hodin do p). Poté M spustí M' na vstup $\langle x, y\rangle$. Stroj M má požadované vlastnosti, a tedy $L \in (\boxed{\exists} \to T)$-$\boldsymbol{P}$.

Je-li naopak $L \in (\boxed{\exists} \to T)$-$\boldsymbol{P}$, pak existuje polynom p a $(\boxed{\exists} \to T)$-TS M přijímající L v čase p. Nechť y popisuje větev výpočtu končící přechodem do stavu, jenž je zobrazen funkcí τ do zdroje typu T. Snadno vytvoříme T-TS M' přijímající v čase $O(p(|x|))$ slovo $\langle x, y\rangle$, právě když y je taková větev vedoucí k přijetí vstupu x. Jazyk L' přijímaný strojem M' patří do T-$\boldsymbol{P}$. Dále $x \in L \Leftrightarrow \exists\, y\,|y| \leq p(|x|) \wedge$ $\wedge \langle x, y\rangle \in L'$, tedy $L \in \exists\,(T$-$\boldsymbol{P})$. □

Důsledek 11.6.1 *Platí* $\Sigma_k^{(3)} = \Sigma_k^{(4)}$ a $\Pi_k^{(3)} = \Pi_k^{(4)}$.

Důkaz: Indukcí: Označme $\sigma_0 = \pi_0 = \boxed{\ }$. Platí $\Sigma_0^{(3)} = \Pi_0^{(3)} = \boxed{\ }\text{-}\boldsymbol{P} = \boldsymbol{P} = \Sigma_0^{(4)} = \Pi_0^{(4)}$.

Vzhledem k tomu, že typy σ_k a $\sigma_k \to \boxed{\ }$ jsou ekvivalentní, pro $k \geq 0$ platí $\Sigma_{k+1}^{(3)} = (\boxed{\exists} \to \pi_k)\text{-}\boldsymbol{P} = \exists\,(\pi_k\text{-}\boldsymbol{P}) = \exists\,\Pi_k^{(3)} \overset{\text{IP}}{=} \exists\,\Pi_k^{(4)} = \Sigma_{k+1}^{(4)}$.

Obdobně vzhledem k tomu, že typy π_k a $\pi_k \to \boxed{\ }$ jsou ekvivalentní, pro $k \geq 0$ platí $\Pi_{k+1}^{(3)} = (\boxed{\forall} \to \sigma_k)\text{-}\boldsymbol{P} = \exists\,(\sigma_k\text{-}\boldsymbol{P}) = \exists\,\Sigma_k^{(3)} \overset{\text{IP}}{=} \exists\,\Sigma_k^{(4)} = \Pi_{k+1}^{(4)}$. □

Věta 11.7 *Platí* $\boldsymbol{PH} \subseteq \boldsymbol{PSpace}$.

Důkaz: Podle tvrzení 9.2.1 je $\boxed{\forall\exists?}\text{-}\boldsymbol{P} \subseteq \boldsymbol{PSpace}$. □

12 Ukázky $\mathcal{T}$-m-Poly-úplných problémů (30. září 1997)

Definice 12.1 Pro libovolné orákulum A a typ T označme

$K(A,T) = \{\langle M, x, 1^t\rangle \mid M$je T-$TS(A)$ přijímající x v čase nejvýš t.

Věta 12.1 *Rozhodnout, zda $\langle M, x, 1^t\rangle \in K(A,T)$, je m-úplný problém pro T-$\boldsymbol{P}(A)$.*

Důkaz: Abychom ukázali $K(A,T) \in T$-$\boldsymbol{P}(A)$, stačí pro dané M, x, t spustit simulaci na universálním Turingově stroji pro typ T. Čas výpočtu bude $O(t \log t)$, což je polynomiální v $|M, x, 1^t|$.

Abychom ukázali T-$\boldsymbol{P}(A)$-m-Poly-těžkost množiny $K(A,T)$, stačí pro libovolný T-$TS(A)$ M, přijímající v polynomiálním čase $p(|x|)$, ukázat transformaci na $K(A,T)$. Touto transformací je funkce $x \to \langle M, x, 1^{p(|x|)}\rangle$. Tuto funkci jsme schopni v polynomiálním čase spočítat. □

12.1 Přirozené m-úplné problémy (formule)

$$Config(\alpha)$$
$$Next(\alpha, \beta)$$
$$Equal(\alpha, \beta)$$
$$Initial(\alpha, x)$$
$$Accepts(\alpha)$$

Věta 12.2 ***SAT****(Splnitelnost booleovské formule v obecném tvaru) je* ***NP****-m-Poly-úplný problém.*

Důkaz: Zřejmě **SAT** $\in$ ***NP***.

Nechť L je jazyk z ***NP***. Nechť $\boxed{\exists}$-TS M přijímá jazyk L v čase $p(n)$. Podle M sestrojme formuli

$$Accepted(x) = \bigwedge_{i=1}^{p(|x|)} Config(\alpha_i) \bigwedge_{i=1}^{p(|x|)-1} Next(\alpha_i, \alpha_{i+1}) \wedge$$
$$\wedge Initial(\alpha_1, x) \wedge Accepts(\alpha_{p_n}).$$

Tato formule je splnitelná, právě když existuje přijímací výpočet stroje M délky $p(n)$ na vstupu x. □

Poznámka 12.1 Počet pravdivých ohodnocení formule $Accepted(x)$ je roven počtu přijímacích výpočtů délky $p(n)$.

Věta 12.3 ***QBF****(Kvantifikované booleovské formule v obecném tvaru) je* ***PSpace****-m-Poly-úplný problém.*

Důkaz: ***PSpace***-m-Poly-těžkost dokážeme nejdříve:

Nechť L je jazyk z třídy ***PSpace*** a nechť TS M přijímá L v prostoru $p(n)$. Počet konfigurací TS M v prostoru $p(n)$ můžeme odhadnout pomocí $2^{c_M \cdot p(n)}$. Vytvoříme kvantifikovanou formuli, která je pravdivá, právě když pro vstup x existuje přijímací výpočet TS M délky nejvýš $2^{c_M \cdot p(|x|)}$.

Vytvoříme postupně formule $Access\_2^m(\alpha, \beta)$, které jsou pravdivé, pokud existuje výpočet TS M přecházející z konfigurace α do konfigurace β v čase 2^m.

$$Access\_2^0(\alpha, \beta) = Config(\alpha) \wedge Config(\beta) \wedge$$
$$\wedge (Equal(\alpha, \beta) \vee Next(\alpha, \beta))$$

Přirozená definice

$$Access\_2^m(\alpha, \beta) = \exists\gamma(Access\_2^{m-1}(\alpha, \gamma) \wedge Access\_2^{m-1}(\gamma, \beta))$$

by bohužel vedla k formuli obsahující 2^m podformulí $Access\_2^0$.

Abychom se tomuto vyhnuli, vynutíme si „opětné použití stejného prostoru“ následujícím trikem:

$$Access\_2^m(\alpha, \beta) = \exists\gamma\forall\alpha', \beta'\big(((Equal(\alpha', \alpha) \wedge Equal(\beta', \gamma)) \vee$$
$$\vee (Equal(\alpha', \gamma) \wedge Equal(\beta', \beta))) \Rightarrow Access\_2^{m-1}(\alpha', \beta')\big)$$

Formule

$$Accepted(x) = \exists\alpha, \beta(Initial(\alpha, x) \wedge$$
$$\wedge Accepts(\beta) \wedge Access\_2^{c_M \cdot p(|x|)}(\alpha, \beta))$$

je pravdivá, právě když TS M přijímá slovo x.

Abychom dokončili důkaz ***PSpace***-m-Poly-úplnosti, ukážeme ještě příslušnost do třídy ***PSpace***:

```
function EvalQBF(F);
begin
  case F of
    1 : return (true);
    0 : return (false);
    ¬F1: return (not EvalQBF(F1));
    F0 ∨ F1: begin
               B0 := EvalQBF(F0);
               B1 := EvalQBF(F1);
               return (B0 ∨ B1)
             end ;
    F0 ∧ F1: begin
               B0 := EvalQBF(F0);
               B1 := EvalQBF(F1);
               return (B0 ∧ B1)
             end ;
    ∃xF1: begin
            B0 := EvalQBF(F1|x:=0);
            B1 := EvalQBF(F1|x:=1);
            return (B0 ∨ B1)
          end ;
    ∀xF1: begin
            B0 := EvalQBF(F1|x:=0);
            B1 := EvalQBF(F1|x:=1);
            return (B0 ∧ B1)
          end
  end
end
```

Alg. 3: Algoritmus vyhodnocující QBF

Algoritmus Alg. 3 vyhodnocuje QBF. Je-li m délka formule, pak hloubka rekurze je nejvýš m. Lokální proměnné jsou velikosti $O(m)$. Algoritmus tedy pracuje v prostoru $O(m^2)$. □

Poznámka 12.2 Chceme-li minimalizovat počet univerzálně kvantifikovaných proměnných, můžeme pro $Access\_2^m(\alpha,\beta)$ použít tvar

$$\exists\gamma\forall z\exists\alpha',\beta'\big((z\Rightarrow(Equal(\alpha',\alpha)\wedge Equal(\beta',\gamma))\wedge$$

$$\wedge(\neg z\Rightarrow Equal(\alpha',\gamma)\wedge Equal(\beta',\beta)))\wedge Access\_2^{m-1}(\alpha',\beta')\big)$$

Poznámka 12.3 Formuli $Access\_2^m(\alpha,\beta)$ můžeme přepsat tak, aby všechny kvantifikátory byly na začátku formule. Do takového tvaru můžeme přepsat i formuli $Accepted(x)$.

Důsledek 12.3.1 ***QBF*** *v prenexním tvaru je* ***PSpace****-m-Poly-úplný problém* □

MAJ je jazyk obsahující logické formule, které jsou splněny při (nadpoloviční) většině ohodnocení prvoformulí.

Je snadno vidět, že uvedený jazyk patří do ***PP***, protože můžeme vytvořit $\boxed{?}$-*TS*, který nejprve náhodně vybere ohodnocení prvoformulí (všechna se stejnou pravděpodobností), a poté vyhodnotí formuli. Podle výsledku přejde do stavu *ANO* nebo *NE*.

Lemma 12.4 *Rozhodovací problém $\#SAT \geq k$ (Počet pravdivých ohodnocení booleovské formule) je* ***PP****-m-Poly-těžký.*

Důkaz: Nechť $A \in \boldsymbol{PP}$, nechť $\boxed{?}$-*TS* M přijímá A. V minulé kapitole jsme ukazovali, jak zkonstruovat logickou formuli $Accepts(x)$, která má právě tolik „dobrých“ ohodnocení, kolik je přijímacích výpočtů TS. Upravme tedy nejprve $\boxed{?}$-*TS* M na $\boxed{?}$-*TS* M' tak, aby v něm každý stav měl právě dva následníky. (Toho můžeme dosáhnout zavedením dvojníků stavů konečné jednotky.) Tím jsme dosáhli toho, že všechny výpočty mají stejnou pravděpodobnost.

Nyní vytvoříme formuli $Accepts(x)$, která má tolik dobrých ohodnocení, kolik je přijímacích větví výpočetního stromu $\boxed{?}$-*TS* M' na vstupu x.

Přijímá–li $\boxed{?}$-*TS* M jazyk A v čase t, pak

$$x \in A \Leftrightarrow \#Accepts(x) \geq 2^{t-1}.$$

Vzhledem k tomu, že $Accepts(x)$ má polynomiální délku vzhledem k $|x|$, ukázali jsme správnou transformaci. □

Věta 12.5 *MAJ i $\#SAT\geq$ k jsou* ***PP****-m-Poly-úplné problémy.*

Důkaz: Vzhledem k tomu, že již víme, že $MAJ \in \boldsymbol{PP}$, a že $\#SAT \geq k$ je ***PP***-těžký, stačí nalézt transformaci problému $\#SAT \geq k$ na MAJ.

Nechť k, F jsou vstupy problému $\#SAT \geq k$, nechť formule F má m proměnných.

Zkonstruujme polynomiálně dlouhou formuli G s týmiž proměnnými, která má přesně $2^m - k$ „dobrých“ ohodnocení. Vytvořme formuli $H = (y \wedge F) \vee (\neg y \wedge G)$, kde y je nová proměnná.

Platí

$$H \in MAJ \Leftrightarrow F \in \#SAT \geq k.$$

□

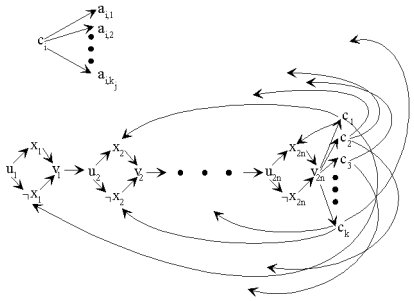
12.2 Další PSpace-m-Poly-úplné problémy

Problém: Je dán orientovaný graf a určen vrchol v_0. Jsou dána následující pravidla hry: Hru hrají dva hráči, kteří střídavě pokládají na vrcholy grafu kamínky, přičemž mohou kamínek položit jedině na neobsazený vrchol do něhož vede hrana z vrcholu na nějž byl naposledy položen kamínek. Na začátku byl položen kamínek právě na vrchol v_0. Hráč který nemůže položit kamínek prohrává. Otázka: Existuje vítězná strategie pro začínajícího hráče?

Věta 12.6 *Právě zformulovaný problém je PSpace-m-Poly-úplný.*

Důkaz: Ukázat, že problém patří do PSpace, je jednoduché. Stačí provést „minimaxové“ prohledání všech herních variant. Stačí si uvědomit, že hloubka rekurze je nejvýš počet vrcholů grafu, protože v každém kroku ubyde počet neobsazených vrcholů.

Zaměříme se na důkaz, že problém je PSpace-m-Poly-těžký. Ukážeme, jak ze zadání problému střídavě kvantifikované formule v polynomiálním čase vytvoříme ekvivalentní zadání našeho problému viz obr. 3. □

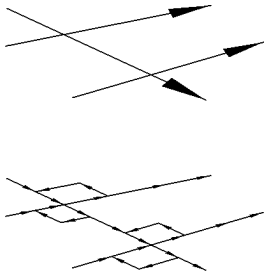


Obr. 3: Převod kvantifikované formule na grafovou hru

Věta 12.7 *Problém existence vítězné strategie v grafové hře je PSpace-m-Poly-úplný i v případě, kdy předem víme, že grafy, na nichž se hraje, jsou rovinné.*

Důkaz: Na obrázku 4 je naznačeno, jak je možno pozměnit konstrukci grafu z obrázku 3, aby byl graf rovinný a aby existence vítězné strategie prvního hráče stále korespondovala s pravdivostí střídavě kvantifikované formule. □

Problém: Je dán orientovaný graf a v něm vyznačeny dva vrcholy v_0, v_1, dále je dáno k kamínků. Jsou dána následující pravidla hry: Hru hraje jediný hráč. Hráč může



Obr. 4: Konec převodu kvantifikované formule na rovinnou grafovou hru

kdykoli sebrat kamínek z libovolného vrcholu grafu. Položit kamínek na vrchol v může jen tehdy, pokud jsou kamínky položeny na všechny vrcholy, z nichž do v vede hrana. Na začátku byl položen kamínek právě na vrchol v_0. Otázkou je, zda může hráč položit kamínek na vrchol v_1.

Není těžké ukázat příslušnost problému do ***PSpace***. Jak je to s těžkostí?

Věta 12.8 *Následující problém je* ***PSpace****-m-Poly-úplný: Problém: Je dána monotónní kontextová gramatika G a slovo x. Otázkou je, zda G generuje x.*

Důkaz: Těžkost je možno dokázat transformací libovolného stroje M s jednoznačnou přijímací konfigurací, s jedinou, jednostrannou páskou, pracující přesně v prostoru $p(|y|)$ pro vstup y. Vytvoříme následující gramatiku:

Každému písmenu páskové abecedy odpovídá terminál. Každému stavu, který není koncový, odpovídá neterminál. Koncovým stavům odpovídají terminály. Stavy se vyskytují na levé straně pouze v pravidlech, které odpovídají přechodové funkci.

$(sQ \to s'Q' \mid \delta(Q,s) = (Q', s', \bullet)$, pro každé písmeno z pravidlo $sQz \to s'zQ' \mid \delta(Q,s) = (Q', s', \to)$, a $sQ \to Q's' \mid$ $\mid \delta(Q,s) = (Q', s', \leftarrow))$

Dále je použit speciální terminál # pro konec použitého úseku pásky. Konfigurace jsou kódovány slovem obsahujícím obsah pásky a ukončeným znakem konce pásky. Dovnitř slova je vložen (ne)terminál označující stav turingova stroje. Tento symbol je vložen za políčko pásky, na němž se nachází hlava.

Počáteční pravidlo vygeneruje pro vstup y stroje M počáteční konfiguraci $S \to y_0, Q_0, y_1, y_2, \ldots, y_{|y|}\#$. Kromě pravidel odpovídajících přechodové funkci, kde je levá strana stejně dlouhá jako pravá, je v G pro každý neterminál odpovídající stavu také pravidlo vkládající symbol blank mezi neterminál a bezprostředně sousedící konec pásky $(Q\# \to Qb\#)$.

Slovo x bude odpovídat jednoznačné přijímací konfiguraci stroje M v prostoru $p(|y|)$.

Ukázat příslušnost problému k ***NPSpace*** je jednoduché. Nedeterministicky volíme pravidla, a ve chvíli, kdy by slovo mělo být delší než $|x|$ zamítáme. Pokud je slovo rovno x, přijímáme. Vzhledem k Savitchově větě patří daný problém do ***PSpace***. □

13 Třída #P, #P–úplné úlohy (12. srpna 1997)

V předchozích přednáškách jsme se zabývali rozhodovacími problémy, třídou **NP**. Připomeňme definici: Formálně můžeme třídu NP definovat jako třídu jazyků vyjádřitelných předpisem

$$\mathcal{L} = \left\{ x \mid \exists^P y \; R^P(x,y) \right\}. \tag{1}$$

Aby byl zřejmý význam symbolů $\exists^P$ a R^P, popíšeme tento předpis naprosto přesně: Pro dané dva polynomy $P_1(n)$, $P_2(n)$ a predikát $R(x,y)$ vyčíslitelný v čase $P_2(|x|+|y|)$ je $\mathcal{L}$ jazyk takových x, pro něž existuje „ověření“ y, $|y| \leq \leq P_1(|x|)$, pro nějž je hodnota predikátu $R(x,y)$ pravda.

O třídě co–NP jsme dosud příliš nemluvili. co–NP znamená “complement nondeterministic polynomial”. Do této třídy patří problémy lišící se od NP–problémů pouze znegováním otázky. Mezi typické zástupce co–NP problémů patří například problémy:

1 Je pravda, že v daném grafu neexistuje klika dané velikosti k?

2 Jsou nutné 4 barvy na obarvení daného rovinného grafu?

Formálně můžeme třídu co–NP definovat jako třídu jazyků vyjádřitelných předpisem

$$\mathcal{L} = \left\{ x \mid \forall^P y \; R^P(x,y) \right\} \tag{2}$$

Samozřejmě za předpokladu „P=NP“ bychom uměli stejně rychle hledat pozitivní i negativní odpovědi. Proto by bylo „NP=P=coNP“.

Třída **#P** je třída úloh, jejichž výsledkem je přirozené číslo. Tuto úlohu bychom mohli formálně definovat takto: Cílem je spočítat pro daný vstup x

$$V_x = \left| \left\{ y \mid |y| \leq P_1(|x|) \mid R^P(x,y) \right\} \right| \tag{3}$$

Neboli pro dané dva polynomy $P_1(n)$, $P_2(n)$, a predikát $R(x,y)$ vyčíslitelný v čase $P_2(|x|+|y|)$ je úlohou pro vstup x určit, kolik existuje „ověření“ y, $|y| \leq P_1(|x|)$, pro nějž je hodnota predikátu $R(x,y)$ pravda.

Na první pohled je vidět, že známe–li V_x — počet ověření predikátu R, umíme rozhodnout rozhodovací problémy „$V_x > 0$?“, „$V_x = 2^{P_1(|x|)} + 2^{P_1(|x|)-1} + \ldots$?“. Jinými slovy umíme řešit rozhodovací problémy rozhodující o příslušnosti x do jazyků (1), (2).

I pro třídu #P se snažíme nalézt „nejtěžší“ úlohy této třídy. Vzhledem k tomu, že hledáme počet ověření, nemůžeme používat polynomiální transformace resp. redukce, které mění neznámým způsobem počet ověření. Můžeme používat takové transformace, které počet ověření nemění—„parsimonious“, a transformace, u nichž známe funkci popisující vztah mezi počty ověření.

Ukážeme, že početní verze problému kachličkování je #P–úplná.

Věta 13.1 *Spočítat počet různých vykachlíčkování koupelny (podle dříve popsaných pravidel) je #P-m-těžká úloha.*

Důkaz: Věta ?? ukazuje, že pro polynom P_1 a polynomiálně vypočítatelný predikát $R(x,y)$ existuje nedeterministický Turingův stroj pracující v polynomiálním čase, jehož počet přijímacích výpočtů je roven počtu ověření $y, |y| \leq \leq P_1(|x|)$. Mějme nedeterministický Turingův stroj těchto vlastností. Nechť má jednoznačnou koncovou konfiguraci, v níž deterministicy setrvává. Stačí si uvědomit, že v důkazu věty ?? jsme zkonstruovali koupelnu a kachlíčky tak, že správné vykachlíčkování koupelny jednoznačně odpovídá jednomu přijímacímu výpočtu NTS. □

Poznámka 13.1 Uvědomme si co znamená x a co znamená y: x je zadání úlohy, tedy tvar a obarvení stěn koupelny a katalog kachlíčků, y je výběr kachlíčků pro jednotlivá políčka.

Věta 13.2 *#SAT je #P-m-úplná úloha. (Počet ohodnocení prvotních formulí, pro něž je formule v konjunktivně disjunktivním tvaru splněna.)*

Důkaz: Při důkazu věty ?? jsme použili „parsimonious“ transformaci z problému kachlíčkování. □

Poznámka 13.2 Protože #SAT $\geq k$ je ***PP***-úplný problém, je $\boldsymbol{\#P \subseteq PF(PP)}$. (Půlením intervalu vyřešíme #SAT.)

Věta 13.3 *#3–SAT je #P-m-úplná úloha.*

Důkaz: Ukážeme, jak pomocí #3–SAT vyřešíme #SAT. Postupně nahradíme každou disjunkci aspoň 4 formulí

$$a_1 \vee a_2 \vee a_3 \vee \cdots \vee a_k$$

konjunkcí s novou prvoformulí x

$$(a_1 \vee a_2 \vee x) \wedge (\neg a_1 \vee \neg x) \wedge (\neg a_2 \vee \neg x) \wedge (\neg x \vee a_3 \vee \cdots \vee a_k).$$

Od transformace z kapitoly ?? se tato liší přidáním konjunkce $(\neg a_1 \vee \neg x) \wedge (\neg a_2 \vee \neg x)$, zajišťující, aby hodnota prvoformule x byla určena jednoznačně. K zakončení důkazu je třeba si uvědomit, že formule se celou konstrukcí prodloužila konstanta–krát. □

Věta 13.4 *#k–klik, #k–nezávislých množin, #(n-k) vrcholových pokrytí jsou #P-m-úplné úlohy.*

Důkaz: Počet nezávislých množin velikosti k je stejný jako počet vrcholových pokrytí velikosti $n-k$ (doplněk konkrétní NM je VP). Počet nezávislých množin velikosti k je stejný jako počet klik velikosti k v grafu, kde hrany jsou nahrazeny nehranami (tytéž množiny vrcholů).

Ukážeme, jak pomocí #k–klik spočítat počet řešení 3–SAT $\exists\, x_1, \ldots, x_n \varphi(x_1, \ldots, x_n)$, kde

$$\varphi = \bigwedge_{i=1}^{m} (a_{i,1} \vee a_{i,2} \vee a_{i,3}),$$

pro $a_{i,j}$ gace nebo negace prvoformulí x_l.

Formuli φ můžeme formálně přepsat do tvaru

$$\varphi = \bigwedge_{i=1}^{m} ((a_{i,1}\ a_{i,2}\ a_{i,3}) \vee (a_{i,1}\ a_{i,2}\neg a_{i,3}) \vee$$
$$(a_{i,1}\neg a_{i,2}\ a_{i,3}) \vee (a_{i,1}\neg a_{i,2}\neg a_{i,3}) \vee (\neg a_{i,1}\ a_{i,2}\ a_{i,3}) \vee$$
$$(\neg a_{i,1}\ a_{i,2}\neg a_{i,3}) \vee (\neg a_{i,1}\neg a_{i,2}\ a_{i,3})),$$

kde „trojice“ uvnitř závorek jsou logické součiny.

Vytvoříme graf skládající se z m sedmic vrcholů a n dvojic vrcholů, vrchol i–té sedmice je označen některou „trojicí“ s prvním indexem i. Vrcholy i–té dvojice jsou označeny x_i a $\neg x_i$.

Dva vrcholy jsou spojeny hranou, pokud logický součin jejich „trojic“ není tautologicky false.

Vidíme, že vrcholy uvnitř sedmice nejsou spojeny hranou a $m+n$–klice v uvedeném grafu jednoznačně odpovídá ohodnocení prvoformulí, pro něž je $\varphi = \mathsf{true}$. □

Věta 13.5 *#k–klik, #k–nezávislých množin, #(n-k) vrcholových pokrytí jsou #P-m-úplné úlohy i v případě, kdy víme, že pro jakékoli větší k problém nemá řešení.*

Důkaz: Stačí si uvědomit, že v důkazu vznikaly transformací pouze grafy u nichž víme, že pro větší k úloha řešení nemá. □

Věta 13.6 *Spočítat počet perfektních párování v bipartitním grafu je #P-T-úplná úloha. (Každý vrchol v právě jedné vybrané hraně.)*

Poznámka 13.3 Toto je první příklad, kde nalézt jedno ověření je jednoduché, ale spočítat je všechny je těžké. Předtím uvedené příklady byly těžké už jako rozhodovací problémy.

(Nalézt řešení je jednoduchá aplikace algoritmu na toky v celočíselných sítích.)

Než tuto větu dokážeme, ukážeme si některé jí ekvivalentní úlohy.

Věta 13.7 *Spočítat počet možných rozestavení věží na předvyznačená políčka šachovnice tak, aby se vzájemně neohrožovaly je #P-T-úplná úloha.*

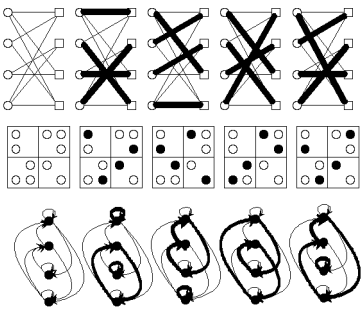
Věta 13.8 *Spočítat počet rozkladů orientovaného grafu na cykly je #P-T-úplná úloha.*

Na obrázku 5 je ukázána vzájemná korespondence mezi párováním, rozmísťováním věží a rozkladem grafu na cykly. (Jedničky na místech matice sousednosti bipartitního grafu (vrcholy jedné partity indexují řádky, vrcholy druhé partity indexují sloupce) označují přípustná políčka na položení věží, tatáž matice slouží jako incidenční matice orientovaného grafu.)

Definice 13.1 Permanent matice $A = (a_{i,j})$ typu $n \times n$ je definován předpisem

$$\mathrm{Perm}\, A = \sum_{\pi \in S_n} \prod_{i=1}^{n} a_{i,\pi(i)}$$

Permanent 0—1 matice je roven počtu možných rozmístění neohrožujících se věží na políčka označená 1. Jiná, ekvivalentní formulace věty 13.6 je věta 13.9:



Obr. 5: Párování, rozmísťování věží a rozklad grafu na cykly

Věta 13.9 *Spočítat permanent 0—1 matice je #P-T-úplná úloha.*

Důkaz věty 13.9 rozdělíme do pěti tvrzení:

Lemma 13.10 *Spočítat počet rozkladů orientovaného grafu na cykly délky větší než 2 je #P-m-úplná úloha.*

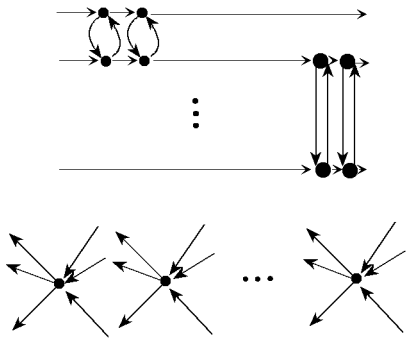
Lemma 13.11 *Spočítat permanent matice s čísly -1, $-\frac{1}{2}$, 0, $\frac{1}{2}$, 1 je #P-m-těžká úloha.*

Lemma 13.12 *Spočítat permanent matice s čísly -2, -1, 0, 1, 2 je #P-m-těžká úloha.*

Lemma 13.13 *Spočítat permanent celočíselné matice modulo součin polynomiálně mnoha prvočísel z počátečního úseku prvočísel je #P-m-těžká úloha.*

Lemma 13.14 *Spočítat permanent celočíselné matice modulo „polynomiálně velké“ prvočíslo je #P-T-těžká úloha.*

Poznámka 13.4 Všimněte si, že v důkazu lemmatu 13.14 se nepoužívá transformace, ale redukce.



Obr. 6: Převod #VP na #orientovaných HK a na #rozkladů na cykly delší než 2

Lemma 13.10 má blízkou souvislost s následující větou.

Věta 13.15 *Spočítat počet Hamiltonovských kružnic je #P-m-úplná úloha nezávisle na tom, zda se jedná o orientovaný nebo neorientovaný graf.*

Důkaz *–13.15*: V kapitole ?? je na obr. ?? uvedena konstrukce, jak vytvořit k danému grafu neorientovaný „graf cest“, v němž výběru k–vrcholového pokrytí ($k-1$ vrcholové pokrytí neexistuje) odpovídá výběr cest. Vybrané cesty můžeme mnoha způsoby spojit v hamiltonovskou kružnici. Možností, jak vybrané cesty spojit do hamiltonovské kružnice, je $2^{k-1}k!(k-1)!$. (Zafixujme jednu cestu, máme $k!$ pořadí výběrů pomocných vrcholů, $(k-1)!$ pořadí výběrů ostatních cest a 2^{k-1} orientací ostatních cest.) Ze znalosti počtu Hamiltonovských kružnic bychom uměli zjistit počet k–vrcholových pokrytí (vydělením číslem $2^{k-1}k!(k-1)!$).

Na obrázku 6 je naznačena konstrukce, jak vytvořit orientovaný „graf cest“, v němž výběru k–vrcholového pokrytí ($k-1$ vrcholové pokrytí neexistuje) opět odpovídá výběr cest. Možností, jak vybrané cesty spojit v hamiltonovskou kružnici, je nyní $k!(k-1)!$, protože orientace je již jednoznačně určena. (Počet vrcholových pokrytí získáme vydělením číslem $k!(k-1)!$.) □

Důkaz *–13.10*: Pokud chceme uvedený orientovaný „graf cest“ rozložit na cykly delší než 2, je opět jednoznačná korespondence mezi „vybranými“ cestami a vybraným k–vrcholovým pokrytím (pokud $k-1$ vrcholové pokrytí neexistuje). Možností, jak vybrané cesty doplnit na cykly, je $(k!)^2$ (můžeme cestám přiřadit libovolné pořadí vstupních a výstupních pomocných vrcholů). (Počet vrcholových pokrytí získáme vydělením číslem $(k!)^2$.) □

Důkaz *–13.11*: Ukážeme, jak pomocí permanentu matice s čísly -1, $-\frac{1}{2}$, 0, $\frac{1}{2}$, 1 počítat počet rozkladů orientovaného „grafu cest“ na cykly delší než 2.

Vezmeme si matici sousednosti „grafu cest“ a nahradíme každou „hranovou“ podmatici (velikosti 4×4) podmaticí podle níže uvedeného schématu:

$$\begin{pmatrix} \ddots & \vdots & 0 & \vdots & 0 & \vdots \\ 0 & 0 & 1 & 1 & 0 & 0 \\ \cdots & 0 & 0 & 0 & 1 & \cdots \\ 0 & 1 & 0 & 0 & 1 & 0 \\ \cdots & 0 & 1 & 0 & 0 & \cdots \\ \cdots & \vdots & 0 & \vdots & 0 & \ddots \end{pmatrix} \to \begin{pmatrix} \ddots & \vdots & 0 & \vdots & 0 & \vdots \\ 0 & 1 & 1 & -1 & 0 & 0 \\ \cdots & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 & \cdots \\ 0 & 0 & 0 & 0 & 1 & 0 \\ \cdots & 1 & -1 & 0 & 0 & \cdots \\ \cdots & \vdots & 0 & \vdots & 0 & \ddots \end{pmatrix}$$

Je potřeba si uvědomit, že příslušný permanent odpovídá počtu rozkladů na cykly delší než 2. □

Důkaz *–13.12*: Kdybychom uměli spočítat permanent každé -2, -1, 0, 1, 2 – matice, uměli bychom permanent matice s čísly -1, $-\frac{1}{2}$, 0, $\frac{1}{2}$, 1 počítat vynásobením prvků matice dvěma a vydělením výsledku 2^n, kde n je rozměr matice. □

Důkaz *–13.13*: Stačí si uvědomit, že celé číslo, na jehož binární zápis nám stačí polynomiálně mnoho bitů, můžeme shora odhadnout součinem polynomiálně mnoha prvočísel z počátečního úseku prvočísel. (Určitě jich stačí tolik, kolik je bitů.)

Na spočítání celého čísla od -2^k do $+2^k$ můžeme použít modulární aritmetiku modulo součin příslušných prvočísel. Máme spočítat číslo od $-2^n n!$ do $2^n n!$. Přitom $2^n n! < 2^n \cdot n^{O(1)} \cdot \frac{n^n}{e^n} < 2^n \cdot 2^{n\log n} < 2^{n^3}$. Uměli bychom spočítat permanent -2, -1, 0, 1, 2 – matice. □

Důkaz *–13.14*: Ukážeme, jak spočítat permanent modulo součin polynomiálně mnoho prvočísel z počátečního úseku prvočísel redukcí na výpočet modulo jednotlivá prvočísla.

Potřebujeme si uvědomit, že n–té prvočíslo je polynomiálně velké vůči n. To plyne z hustoty prvočísel $\frac{c}{\log n}$. (Do n^2 je asi $\frac{cn^2}{\log n} > n$ prvočísel.)

Dále je třeba si uvědomit, že ze znalosti výsledku modulo jednotlivá prvočísla jsme schopni rychle spočítat výsledek modulo jejich součin. Pracujeme s aritmetikou potřebující polynomiální počet bitů!

Nechť $q \equiv q_i \pmod{p_i}$ a nechť $r_i \cdot \prod_{j\neq i} p_j \equiv 1 \pmod{p_i}$. Potom

$$q \equiv \sum q_i \cdot r_i \prod_{j\neq i} p_j \pmod{\prod p_i}.$$

(K ověření vztahu stačí zkontrolovat, že

$$q_i \equiv \sum q_k \cdot r_k \prod_{j\neq k} p_j \pmod{p_i},$$

protože vektor modul jednotlivá prvočísla je jednoznačný.)
□

Důkaz *–13.9*: Ukážeme, jak pomocí permanentu z 0–1 matice spočítat permanent matice s malými celými nezápornými čísly ($< p$). Celkový součet čísel v matici je nejvýš pn^2. Místo každého čísla k většího než 1 vložíme k řádků a sloupců podle následujícího schématu (kde $k=3$):

$$\begin{pmatrix} & \cdots & b & \cdots \\ \vdots & \ddots & \vdots & \vdots \\ a & \cdots & 3 & \cdots \\ \vdots & \cdots & \vdots & \ddots \end{pmatrix} \to \begin{pmatrix} & \cdots & b & s_1 & s_2 & s_3 & \cdots \\ \vdots & \ddots & \vdots & 0 & 0 & 0 & \cdots \\ a & \cdots & 0 & 1 & 1 & 1 & \cdots \\ r_1 & 0 & 1 & 1 & 0 & 0 & 0 \\ r_2 & 0 & 1 & 0 & 1 & 0 & 0 \\ r_3 & 0 & 1 & 0 & 0 & 1 & 0 \\ \vdots & \cdots & \vdots & 0 & 0 & 0 & \ddots \end{pmatrix}$$

Vytvořili jsme 0–1 matici s nejvýš $n+pn^2$ řádky a sloupci, jejíž permanent je stejný jako permanent původní matice.
□

14 Důvěřuj, ale prověřuj (2. června 1998)

V této kapitole se budeme zabývat jazyky, u kterých je možno (s dostatečnou pravděpodobností) přesvědčit ověřovatele pracujícího v polynomiálním čase o náležení slova do jazyka. V případě nenáležení do jazyka na to ověřovatel s dostatečnou pravděpodobností příjde.

Problémy třídy ***NP*** je možno jednoduše (pozitivně) odpovědět, pokud nám někdo napoví hledané řešení. Pokud je odpověď negativní, nemůže nám nikdo hledané řešení ukázat. Pro problémy třídy ***NP*** tedy existuje jednoduchý protokol, v němž ověřovatel pracuje deterministicky v polynomiálním čase.

14.1 $\boxed{\exists?}$-*TS* a interaktivní protokoly

Má-li ověřovatel dobrý generátor náhodných čísel, může prověřit příslušnost k mnohem větším jazykům. V takovém případě ale nemůže mít stoprocentní jistotu.

Jednou možností protokolu je protokol mezi „ověřovatelem" a jedním „dokazovatelem". Cílem ověřovatele je s dostatečnou pravděpodobností odhalit jakýkoli případný podvod podvádějícího dokazovatele. Podvádějící dokazovatel si může pamatovat celý protokol, proto ve skutečnosti neodpovídá na jednotlivé dotazy, ale jeho odpověď může být závislá na celém průběhu protokolu.

Vzhledem k tomu, že ověřovatel má pouze polynomiální čas, jsou velikosti správ vyměňovaných mezi ověřovatelem a dokazovatelem omezeny polynomem. Protože ověřovatel používá náhodný generátor, zajímá nás pro každé slovo w pravděpodobnost jeho přijetí $P(O,D)(w)$. (Výsledek výpočtu samozřejmě závisí také na dokazovateli.)

Definice 14.1 Říkáme, že jazyk L má *interaktivní protokol*, píšeme $L \in \boldsymbol{IP}$, pokud existuje ověřovatel O takový, že

$$\exists\, D\, \forall\, w \in L \quad P(O,D)(w) > 2/3$$

a

$$\forall\, D\, \forall\, w \notin L \quad P(O,D)(w) < 1/3.$$

Poznámka 14.1 Konstanty $p_p = 2/3, p_z = 1/3$ v definici ***IP*** nejsou podstatné, důležité je pouze to, že $p_p > p_z$. Stejně jako u třídy ***BPP*** jsme schopni pravděpodobnosti iterováním vylepšovat. Dokonce jsme schopni provést tuto „iteraci" tak, že se nezvýší počet interakcí mezi ověřovatelem a dokazovatelem. V každé interakci může být totiž posláno „paralelně" několik zpráv. Můžeme proto „iterovat" jakoby „paralelně".

Příklad 14.1 Příkladem je protokol ověřující, že grafy G_0, G_1 nejsou izomorfní: Ověřovatel za i náhodně zvolí buď 0 nebo 1, vytvoří graf G náhodnou permutací vrcholů grafu G_i a zeptá se dokazovatele, který z grafů G_0, G_1 je s grafem G izomorfní. Pokud dokazovatel oznámí graf G_i, ověřovatel přijme.

Pokud jsou grafy G_0, G_1 izomorfní, dokazovatel nemá žádnou možnost, jak zjistit volbu čísla i. Ať odpoví jakkoli, pravděpodobnost, že zvolí stejné i jako ověřovatel, je 1/2 ($\forall\, x \notin L\, \forall\, D\, P(O,D)(x) \leq 1/2$). Naopak dokazovatel, který umí testovat izomorfismus, může vždy v případě neizomorfismu grafů G_0, G_1 určit, který (jediný) z nich se rovná G ($\exists\, D\, \forall\, x \in L\, P(O,D)(x) = 1$).

Poznámka 14.2 Tak jako u pravděpodobnostních tříd i zde omezíme možnost náhodného větvení v každém náhodném stavu ověřovatele na dvě větve se stejnou pravděpodobností. Jiné druhy větvení (s konstantními pravděpodobnostmi) bychom v našem modelu mohli aproximovat. (Vzhledem k tomu, že mezi pravděpodobností přijetí a nepřijetí je celý interval hodnot, aproximací pouze tento interval zmenšíme.)

Jemnějším dělením třídy ***IP*** je rozdělení ***IP*** na třídy ***IP[$f(n)$]***, kde funkce $f(n)$ udává maximální možný počet interakcí mezi ověřovatelem a dokazovatelem. Neizomorfismus grafů patří do třídy ***IP[2]***.

Podle příkladu protokolu na neizomorfismus grafů by se mohlo zdát, že důležitým prvkem v protokolu je možnost zatajení náhodného rozhodnutí ověřovatele. Této možnosti je zabráněno v hrách Artuše s Merlinem.

Poznámka 14.3 Anglicky se používá názvu Arthur-Merlin game/protocol. Podle mytologie byl Merlin rádce krále Artuše, proto správný překlad je hra/protokol Artuše (nikoli Artura) s Merlinem.

Definice 14.2 Řekneme, že interaktivní protokol je *hra/protokol Artuše s Merlinem*, pokud ověřovatel (Artuš) zasílá všechny náhodné bity Merlinovi.

Definice 14.3 Třída ***AM*** je třída jazyků, pro něž existuje interaktivní protokol, který je hrou Artuše s Merlinem.

Poznámka 14.4 Když zde hovoříme o hře, měl bych upozornit na to, že obecně je vhodné pohlížet na interaktivní protokol jako na hru mezi ověřovatelem a dokazovatelem. Z tohoto úhlu pohledu můžeme definovat optimálního dokazovatele jako dokazovatele s nejlepší strategií. Je to dokazovatel, který ověřovatele přesvědčí o příslušnosti do jazyka s největší pravděpodobností. Podle definice optimální dokazovatel přesvědčí ověřovalete s pravděpodobností větší než 2/3, patří-li vstup do jazyka, a s pravděpodobností menší než 1/3, pokud vstup do jazyka nepatří.

Obdobně jako pro třídu ***IP*** definujeme třídy ***AM[$f(n)$]*** omezením ***AM*** na základě počtu interakcí mezi Artušem a Merlinem.

Poznámka 14.5 Vzhledem k tomu, že Artuš oznamuje Merlinovi všechna svá náhodná rozhodnutí, nemusí mu již oznamovat nic jiného. Merlin si ostatní může dopočítat. (Zná Artušův program.) Odtud je již velmi malý krůček k charakterizaci ***AM*** jako třídy rozpoznávané v polynomiálním čase $\boxed{\exists?}$-*TS* ve smyslu BPP.

Věta 14.1 ***AM*** *je třída jazyků L rozpoznávaných* $\boxed{\exists?}$*-TS v polynomiálním čase, kde navíc $E(x) \leq \varepsilon < \frac{1}{2}$.*

Důkaz: Program Artuše je možno proložit existenčními stavy, které slouží k vygenerování odpovědi optimálního dokazovatele.

Opačně, máme-li $\boxed{\exists?}$-*TS*, můžeme z něj udělat protokol tak, že Merlin bude oznamovat, jak má program pokračovat v existenčních stavech, abychom měli největší pravděpodobnost přijetí. □

Věta 14.2 $\boldsymbol{IP = AM}$.

Důkaz: Inkluze $\boldsymbol{AM \subseteq IP}$ plyne z toho, že hry Artuše s Merlinem jsou speciálním případem interaktivních protokolů.

Nechť $L \in \boldsymbol{IP}$. Nechť O_1 je ověřovatel, který toto garantuje. Označme $\ell(n)$ maximální možný počet náhodných bitů použitých ověřovatelem O_1 na vstupy délky n. Změníme protokol tak, že nový ověřovatel O_2 pro vstup délky n vždy vygeneruje právě $\ell(n)$ náhodných bitů a pošle je dokazovateli na konci komunikace (tímto už nijak nebude ovlivněn výsledek výpočtu). Dále změníme protokol tak, aby zprávy vyměňované mezi ověřovatelem O_2 a dokazovatelem byly pouze jednobitové (stačí dosavadní protokol proložit bity, jimž nikdo nevěnuje pozornost).

Vzhledem k tomu, že je možných právě $2^{\ell(n)}$ náhodných průběhů programu ověřovatele O_2, můžeme pravděpodobnost přijetí popsat ve tvaru $k/2^{\ell(n)}$. Průběh protokolu si můžeme představit jako strom možných variant. V lichých krocích ověřovatel náhodně vybírá jednu ze dvou větví jak pokračovat, v sudých krocích vybírá pokračování dokazovatel. (Ověřovatel v průběhu protokolu vygeneruje pouze $\ell(n)$ náhodných bitů, proto je jeho pokračování často vynucené.) Strom končí v hloubce odpovídající délce protokolu. Některé listy jsou označeny jako přijímací.

Je-li stanovena strategie dokazovatele, dostáváme pro daný vstup strom různých výpočtů, v němž jsou větvení pouze v lichých vrcholech (větvení ověřovatele). Vzhledem k tomu, že je právě $2^{\ell(n)}$ náhodných průběhů programu ověřovatele, má tento strom „dokazatelovy strategie“ právě $2^{\ell(n)}$ listů.

Nyní můžeme popsat protokol ověřovatele O_3. Nejdříve se ověřovatel zeptá dokazovatele na počet listů přijímajícího podstromu (pro daný vstup w) dokazovatelovy strategie (vůči O_2). Jestliže odpověď $k(w)$ dokazovatele je nejvýš $\frac{2}{3}2^{\ell(|w|)}$, ověřovatel O_3 slovo w zamítne. Je-li $k(w)$ větší než $\frac{2}{3}2^{\ell(|w|)}$, ověřovatel O_3 vybere číslo větve rovnoměrně od 1 do $k(w)$. A ověří, že i-tá větev dokazovatelem určeného podstromu vede do přijímajícího listu (a také, že větvení tohoto výpočtu odpovídá náhodným bitům specifikovamým na konci protokolu). Pravděpodobnost nalezení větve vedoucí do přijímacího listu bude pak úměrná poměru počtu přijímacích větví určeného podstromu ($\leq$ $\leq \alpha(w)$) vůči velikosti $k(w)$ tohoto podstromu. Pokud $w \in$ $\in L$, pak dokazatel může poslat $k(w) = \alpha(w) > \frac{2}{3}2^{\ell(n)}$ a ověřovatel přijme s pravděpodobností 1. Pokud $w \notin$ $\notin L$, pak je pro libovolného dokazovatele $\alpha(w) < \frac{1}{3}2^{\ell(n)}$, a tedy $\alpha(w)/(\frac{2}{3}2^{\ell(n)}) < 1/2$, takže pravděpodobnost přijetí je nejvýš 1/2.

Ještě je třeba ukázat, jak O_3 rovnoměrně vybere větev dokazatelem určeného podstromu. K tomu stačí zeptat se dokazovatele před každým větvením, kolik větví podstromu pokračuje kterým směrem (přitom O_3 kontroluje, zda součet odpovídá počtu větví celého podstromu), a na základě náhodného generátoru s pravděpodobností ve vzájemném poměru počtu větví obou směrů určit pokračování. Vzhledem k tomu, že jsou obecně potřeba racionální pravděpodobnosti, je potřeba je aproximovat pravděpodobnostmi typu $p/2^q$. Kvůli aproximaci se pravděpodobnost přijetí slova nepatřícího do L nepodstatně zvýší.

Na závěr důkazu je potřeba si uvědomit, že ověřovatel O_3 může svá náhodná rozhodnutí posílat dokazovateli, jedná se tedy o protokol Artuše s Merlinem. $L \in \boldsymbol{AM}$. □

Z možnosti deterministické simulace $\boxed{\exists?}$-*TS* pracujícího v čase t v prostoru $O(t^2)$ (viz věta 9.2) plyne:

Lemma 14.3 $\boldsymbol{AM \subseteq PSpace}$

Věta 14.4 $\boldsymbol{AM = PSpace}$

Než ukážeme protokol Artuše s Merlinem přijímající pravdivé libovolně kvantifikované formule (tedy protokol pro $\boldsymbol{PSpace}$-úplný problém), definujeme aritmetizaci formule.

Definice 14.4 Nechť (kvantifikovaná) formule φ obsahuje z logických spojek pouze konjunkce a disjunkce. Nechť se negace vyskytují pouze u proměnných.

Indukcí podle složitosti takovýchto formulí definujeme aritmetizaci $A(\varphi)$ jakožto polynom ve volných proměnných formule podle následujících pravidel:

$$\begin{aligned}
A(\exists x\varphi) &= \textstyle\sum_{x=0}^{1} A(\varphi),\\
A(\forall x\varphi) &= \textstyle\prod_{x=0}^{1} A(\varphi),\\
A(\varphi_1 \wedge \varphi_2) &= A(\varphi_1)\cdot A(\varphi_2),\\
A(\varphi_1 \vee \varphi_2) &= A(p_1) + A(\varphi_2),\\
A(\neg x) &= 1 - x,\\
A(x) &= x.
\end{aligned}$$

Lemma 14.5 *Je-li pro formuli φ definována aritmetizace, pak touto aritmetizací je polynom ve volných proměnných formule φ. Každá proměnná se v polynomu vyskytuje pouze v mocninách menších než $2^{|\varphi|}$, kde $|\varphi|$ značí délku zápisu formule. Součet absolutních hodnot koeficientů v polynomu $A(\varphi)$ je menší než $2^{2^{|\varphi|}}$.*

Důkaz: Indukcí podle složitosti formule. □

Lemma 14.6 *Je-li pro uzavřenou formuli definována aritmetizace, pak jsou následující podmínky ekvivalentní:*
1. Aritmetizace formule je nenulová.
2. Aritmetizace formule je kladná.
3. Formule je pravdivá.

Důkaz: Indukcí podle složitosti formule. □

Abychom mohli použít aritmetizaci formule, převedeme nejprve formuli do tvaru, v němž jsou všechny logické spojky nahrazeny pomocí konjunkcí disjunkcí a negací, poté převedeme formuli do tvaru, kde se vyskytují negace pouze u proměnných.

Podle lemmat 14.5 a 14.6 stačí k ověření pravdivosti uzavřené formule φ ověřit, že aritmetizace je číslo z intervalu $\langle 1, 2^{2^{|\varphi|}}\rangle$.

Lemma 14.7 *Nechť $A \in \langle 0, 2^{2^N})$, pak $A \neq 0$ právě když existuje prvočíslo $p \in (2^N, 2^{2N})$ takové, že $A \not\equiv 0 \pmod p$.*

Důkaz: Je-li $A = 0$, pak pro libovolné číslo p platí $A \equiv 0 \pmod p$.

Pokud $A \in \langle 1, 2^{2^N})$, pak stačí ukázat, že součin prvočísel z intervalu $(2^N, 2^{2N})$ nedělí A. To ale při dostatečně velkém N plyne z toho, že tento součin je větší než A. Logaritmus čísla A je totiž nejvýš 2^N, Logaritmus každého z uvažovaných prvočísel je aspoň N, a těchto prvočísel je zhruba $\frac{(2^{2N}-2^N)}{\ln 2^N}$ vzhledem k tomu, že hustota prvočísel v okolí n je $\frac{1}{\ln n}$. Logaritmus součinu uvažovaných prvočísel bude víc než $2^N \log e > 2^N$. □

Předchozí lemma nám umožňuje zredukovat velikost čísel, jež je třeba přenášet protokolem z původních čísel s exponenciálně mnoha bity na čísla s polynomiálně dlouhým zápisem. Stále ale není možné v polynomiálním čase předávat aritmetizace podformulí, protože se jedná o polynomy exponenciálních stupňů. Redukovat stupně polynomů můžeme tím, že zvolíme vhodný tvar formule φ.

Pro účely této sekce definujeme „vhodný“ a „vhodněprenexní“ tvar formule a „vhodnokvantifikátor dvojníka“.

Definice 14.5 Formule je ve vhodném tvaru, právě když se v jejím zápisu před posledním výskytem libovolné volné proměnné a mezi kvantifikací a výskytem libovolné vázané proměnné vyskytuje nejvýš jeden všeobecný kvantifikátor.

Zavedeme zkratku $D\, y_i/y_i'$ místo zápisu $\exists\, y_i'\ ((y_i' \wedge y_i) \vee \vee (\neg y_i' \wedge \neg y_i)) \wedge$.

Zápisy $\forall$, $\exists$ a $D\, x_j/$ nazveme vhodnokvantifikátory.

Formule φ je ve vhodněprenexním tvaru, právě když $\varphi = Q_1 x_1 Q_2 x_2 \ldots Q_k x_k \psi(x_1, x_2, \ldots, x_k)$, kde Q_i zastupuje některý ze vhodnokvantifikátorů a ψ neobsahuje žádný kvantifikátor a φ je ve vhodném tvaru.

Lemma 14.8 *Nechť je pro formuli φ vhodného tvaru definována aritmetizace Potom je v aritmetizaci stupeň každé proměnné menší než $2|\varphi|$.*

Důkaz: Indukcí podle složitosti formule. □

Pokud tedy převedeme uzavřenou formuli do vhodného tvaru, pak aritmetizace všech podformulí budou polynomy „malého“ stupně.

Lemma 14.9 *Libovolnou formuli φ můžeme v polynomiálním čase převést na formuli ψ vhodného tvaru se stejnými volnými proměnnými, nabývající stejných hodnot.*

Důkaz: Indukcí podle složitosti formule. Podle indukčního předpokladu nalezneme ψ_i k podformuli φ_i. Jediný problém přináší situace $\varphi = \forall x \varphi_1(x, y_1, y_2, y_3, \ldots, y_k)$, kde $y_1, \ldots, y_k$ jsou volné proměnné formule φ. Tento případ vyřešíme přidáním existenčně kvantifikovaných dvojníků y_i' k proměnným y_i, zajistíme, aby se jednalo o dvojníky a nahradíme každý výskyt proměnné y_i v podformuli φ_1 proměnnou y_i'. Dostáváme tak formuli $\psi = D\, y_1/y_1'\ D\, y_2/y_2'\ \ldots D\, y_k/y_k'\ \psi_1(x, y_1', y_2', \ldots, y_k')$. Snadným cvičením je ověřit, že formule ψ má požadované vlastnosti. □

Postup důkazu předchozího lemmatu převede uzavřenou formuli v prenexním tvaru do vhodněprenexního tvaru.

Stačí proto nalézt protokol pro vhodněprenexní formule.

Důkaz: (Věty) Protokol dokazující pravdivost uzavřené vhodněprenexní formule φ proběhne tak, že dokazovatel (Merlin) pošle nejdříve prvočíslo $2^{|\varphi|} < p < 2^{2|\varphi|}$, pro nějž $k \equiv A(\varphi) \not\equiv 0 \pmod p$.

Existuje ***NP*** protokol dokazující prvočíselnost. Pro naše účely ale stačí, že známe ***BPP*** (dokonce ***co-R***) algoritmus ověřování prvočíselnosti. Artuš proto k ověření prvočíselnosti Merlina vůbec nepotřebuje.

Poté bude probíhat protokol v aritmetice modulo p, dokazující $k \equiv A(\varphi) \pmod p$.

Označme φ_1, φ_2, ..., φ_n podformule formule φ vzniklé postupným odstraňováním jednotlivých kvantifikátorů. Označme $p_i(x_1, \ldots, x_i) \equiv A(\varphi_i) \pmod p$.

Protokol bude probíhat v $n + 1 < |\varphi|$ fázích. V nulté fázi Merlin pošle číslo $k = p_0 \equiv A(\varphi)$. V i–té fázi Merlin pošle polynom $q_i(x_i) \equiv p_i(a_1, \ldots, a_{i-1}, x_i)$, kde a_j je číslo náhodně zvolené Artušem na konci j–té fáze protokolu. Podle lemmatu 14.8 je $q_i(x_i)$ polynom stupně méně než $2|\varphi|$. Artuš tuto skutečnost zkontroluje. Artuš dále podle typu vhodnokvantifikátoru, jemuž odstraněný kvantifikátor odpovídal, porovná $q_{i-1}(a_{i-1})$ buď s výrazem $q_i(0) \cdot q_i(1)$ v případě $\forall$, nebo s výrazem $q_i(0) + q_i(1)$ v případě $\exists$ a nebo s výrazem $(1 - a_j)q_i(0) + a_j q_i(1)$ v případě $D\, x_j/$. Na konci i–té fáze Artuš zvolí náhodně číslo a_i (každé s pravděpodobností $\frac{1}{p}$). Hodnotu $p_n(a_1, \ldots, a_n) \pmod p$ dokáže spočítat Artuš sám. Pokud v průběhu provádění protokolu Artuš neobjeví žádnou nesrovnalost, pak vstup φ přijme.

V každé fázi protokolu jsou předávány polynomy stupně menšího než $2|\varphi|$ v nejvýš $2|\varphi|$-bitové aritmetice. Na komunikaci mezi Artušem a Merlinem nám proto stačí polynomiální čas v $|\varphi|$.

Pokud skutečně $k \equiv A(\varphi)$, pak právě popsaný Merlin přesvědčí Artuše s pravděpodobností 1. (Aritmetizaci $D\, x/y\ \varphi(y) = \exists y\, ((x \wedge y) \vee (\neg x \wedge \neg y)) \wedge \varphi(y)$ odpovídá $\sum_{y=0}^{1}(xy + (1-x)(1-y))A(\varphi)(y) = (1-x)A(\varphi)(0) + x A(\varphi)(1)$.)

Pokud $k \not\equiv A(\varphi)$, nechť je i číslo fáze, v níž se naposledy $q_i^M(x_i)$ posílané Merlinem neshoduje s $q_i(x_i)$ které by měl Merlin poslat kdyby postupoval podle definice v protokolu. Pokud Artuš nenalezl nesrovnalost při kontrole $q_i(a_i)$, potom vzhledem k tomu, že $q_{i+1}(0) = q_{i+1}^M(0)$ a $q_{i+1}(1) = q_{i+1}^M(1)$ je a_i kořen nenulového polynomu $q_i - q_i^M$ stupně menšího než $2|\varphi|$. Pravděpodobnost, že se Artuš v i–té fázi trefí do kořene polynomu $q_i - q_i^M$ je menší než $\frac{2|\varphi|}{p}$. Pravděpodobnost, že existuje fáze, v níž se Artuš trefí do kořene příslušného polynomu je menší než $(n+1) \cdot \frac{2|\varphi|}{p} < \frac{2|\varphi|^2}{2^{|\varphi|}}$. Proto pro libovolného Merlina je pravděpodobnost přijetí nepravdivé formule φ mizivá. □

14.2 *MIP*, orákulum jako dokazovatel

Jinou možností protokolu je protokol mezi „ověřovatelem“ a více nezávislými „dokazovateli“. Dokazovatelé nemohou

navzájem komunikovat. Mají dohodnutou strategii, ale znají pouze průběh své komunikace s ověřovatelem. Dá se ukázat, že na počtu dokazovatelů nezáleží, že stačí, když jsou dva. V takovém případě dokazovatelé neznají celou historii protokolu, čímž je téměř vynuceno, aby dokazovatelé na konkrétní dotaz odpovídali vždy všichni stejně, protože nemohou tušit, zda tentýž dotaz nebude pro kontrolu položen jinému dokazovateli.

Jednou z možných definic třídy ***MIP*** je následující definice:

Definice 14.6 ***MIP*** je třída jazyků L, pro něž existuje ověřovatel O $\boxed{?}$-TS s orákulem, pracující v polynomiálním čase, kde

$$\exists D\,,\ D \text{ je orákulum } \forall w \in L \quad P(O,w)(D) > 2/3$$

a

$$\forall \text{ orákulum } D \forall w \notin L \quad P(O,w)(D) < 1/3.$$

Poznámka 14.6 Zatajování informace je zajištěno tím, že orákulum není funkce závislá na průběhu výpočtu, tedy ani na náhodném generátoru ověřovatele.

To, že zatajování historie protokolu je mnohem důležitější než zatajování náhodných rozhodnutí ověřovatele ukazuje následující věta, kterou uvádím bez důkazu. Důkaz je možno nalézt v literatuře.

Věta 14.10 MIP = NExPTime

Literatura:

[G] S. Goldwasser: "Interactive Proof Systems", *Vol. 38 Proc. of Symp. in Applied Mathematics, 1989 (p. 108-128)*

[LFK] C. Lund, L. Fortnow, H. Karloff: "Algebraic Methods for Interactive Proof Systems", *Proc. 31st FOCS, 1990*

[BFL] L. Babai, L. Fortnow, C. Lund: "Non-Deterministic Exponential Time has Two-Prover Interactive Protocols" *Proc. 31st FOCS, 1990*

15 Booleovské obvody, $P/poly$, Paralelismus (28. května 1998)

V této kapitole se budeme věnovat booleovským obvodům — výpočetnímu modelu, který lze s naším „sekvenčním modelem“ porovnávat jen velmi těžce.

Prvním problémem je, že každý booleovský obvod je určen pouze pro vstupy pevné velikosti. K vyřšešení tohoto problému jsou vhodné „rádcovské funkce“.

Jiný problém tkví v jiném způsobu měření složitosti. U booleovských obvodů je rozumné měřit velikost případně hloubku obvodu. Ukážeme, jak „nárůst velikostí obvodů“ pro daný jazyk souvisí s časem Turingova stroje, a jak „nárůst hloubek obvodů“ souvisí s prostorem Turingova stroje. Poslední tvrzení je velmi podobné „Paralelní tezi“, vzhledem k tomu, že hloubka booleovského obvodu odpovídá času v tomto paralelním modelu výpočtu.

Definice 15.1 Booleovský obvod je acyklický orientovaný graf. Vrcholy s nenulovým vstupním stupněm jsou nazývány hradla a jsou označeny $\vee$, $\wedge$ nebo $\neg$ s tím, že příslušné vrcholy musí mít vstupní stupeň po řadě 2, 2, 1. Ostatní vrcholy (se vstupním stupněm 0) jsou nazývány vstupy obvodu. Ze vstupů jsou dva speciální, jeden má vyhrazenu hodnotu 1(true), druhý hodnotu 0(false). Ostatní vstupy jsou nazývány proměnné obvodu. Proměnné jsou očíslovány.

Funkce $f : V \to \{0,1\}$ je ohodnocením Booleovského obvodu pokud se ohodnocení libovolného hradla shoduje s funkcí naznačenou označením hradla aplikovanou na ohodnocení vrcholů, z nichž vede hrana do daného hradla.

Poznámka 15.1 Většinou nás ani nezajímá ohodnocení všech hradel Booleovského obvodu, ale pouze ohodnocení jediného „výstupního“ hradla.

Je-li n počet proměnných takového obvodu O, pak definujeme jazyk $L_O \subseteq \{0,1\}^n$ přijímaný obvodem, tak, že $x = x_1x_2\ldots x_n \in L_O$ právě když je výstupní hradlo obvodu O pro proměnné x_1, x_2, ..., x_n ohodnoceno true.

Definice 15.2 Nechť L je jazyk nad abecedou $\{0,1\}$. Označme $c_L(n)$ velikost (počet hradel) nejmenšího Booleovského obvodu přijímajícího jazyk $L \cap \{0,1\}^n$. Obdobně označme $d_L(n)$ hloubku (počet hran nejdelší cesty) nejmělčího obvodu přijímajícího jazyk $L \cap \{0,1\}^n$.

Poznámka 15.2 Nemusí existovat obvod přijímající $L \cap \{0,1\}^n$, pro nějž by byla zároveň velikost $c_L(n)$ a hloubka $d_L(n)$.

CVP je problém, který rozhodne, zda daný booleovský obvod přijímá pro zadané ohodnocení proměnných. Booleovský obvod je zadáván ve standardizovaném tvaru. Tímto tvarem je popis obvodu v topologickém pořadí hradel, s tím, že poslední hradlo je hradlem výstupním. V popisu jsou zahrnuta hradla (kromě vstupních), pro každé hradlo h je uveden typ a číslo hradla h a čísla hradel z nichž vedou vstupní hrany do hradla h. (Hradla proměnných jsou číslována od jedné, konstantě false odpovídá hradlo s číslem 0, konstantě true odpovídá hradlo s číslem -1. Čísla hradel odpovídají jejich topologickému pořadí.)

Cvičení 15.1 Dokažte, že vyhodnotit daný booleovský obvod velikosti c pomocí Turingova stroje je možno v čase $c^{O(1)}$ (Neboli **CVP** $\in P$).

Dokažte, že vyhodnotit daný booleovský obvod hloubky d s jediným stokem pomocí Turingova stroje je možno v prostoru $O(d^2)$. (Stok ... hradlo bez výstupních hran.)

15.1 Rádcovské funkce

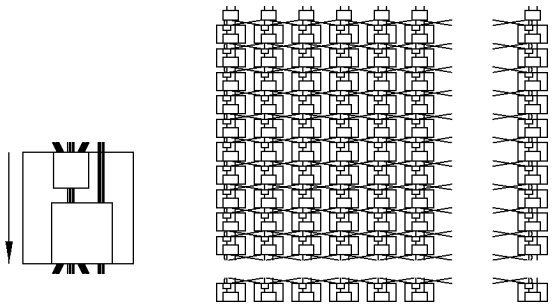
Funkce $\mathbb{N} \to \Sigma^*$ budeme nazývat rádcovské funkce. Nechť log jsou rádcovské funkce, kde $|f(n)| \in O(\log n)$. Nechť $poly$ jsou rádcovské funkce, kde $|f(n)| \in n^{O(1)}$.

Definice 15.3 Nechť $\mathcal{C}$ je třída jazyků a $\mathcal{F}$ třída rádcovských funkcí. Třída $\mathcal{C}/F$ je třída jazyků, kde $B \in \mathcal{C}/F \Leftrightarrow$ $\Leftrightarrow \exists\, A \in \mathcal{C}, f \in \mathcal{F}$ tak, že $B = \{x \mid \langle x, f(|x|)\rangle \in A\}$.

Poznámka 15.3 Evidentně $\boldsymbol{P} \subseteq \boldsymbol{P}/log \subseteq \boldsymbol{P}/poly$.

15.2 Velikost booleovských obvodů

Věta 15.1 *Nechť $L \subseteq \{0,1\}^*$. Pokud deterministický jednopáskový TS přijímá jazyk L v čase $t(n)$ a prostoru $s(n)$, pak $c_L(n) \in O\big(s(n)t(n)\big)$.*



Obr. 7: Nahrazení Turingova stroje Booleovským obvodem

Důkaz: Turingův stroj nejdříve převedeme na stroj s jednostraně (doprava) nekonečnou páskou a s jednoznačnou koncovou konfigurací, končící na začátku pásky ve stavu q_f. V této konfiguraci Turingův stroj setrvává.

Potom na základě přechodové funkce vytvoříme „základní obvod“, který bude mít jako vstup symbol pásky (několik bitů) a stav TS (několik bitů). Kódování stavu TS je takové, že žádný stav není kódován samými nulami a pouze kód stavu q_f obsahuje jedničku v nejlevějším bitu. Výstupem základního obvodu bude symbol pásky a trojice stavů odpovídající třem možným posunům hlavy po pásce. Je-li stavový vstup základního obvodu nulový, jsou všechny stavové výstupy nulové a symbolový výstup je kopií symbolového vstupu. Je-li stavový vstup základního obvodu nenulový, je právě jeden stavový výstup nenulový a společně se symbolovým výstupem vše odpovídá přechodové funkci.

Meziobvod má tři stavové vstupy a jediný stavový výstup. Tento výstup je vytvořen bitovým `or` vstupů.

Rozšířený obvod napojuje základní obvod za meziobvod. Rozšířený obvod má tedy tři stavové vstupy a výstupy a jeden symbolový vstup a výstup.

Booleovský obvod, přijímající $L \cap \{0,1\}^n$, vytoříme pomocí jedné řady po $s(n)$ základních obvodech a z $t(n)$ řad po $s(n)$ rozšířených obvodech. Vstupy meziobvodů jsou odpovídající výstupy tří základních obvodů předchozí řady. Výjimkou jsou krajní obvody, kde jsou pro dva vstupy použity dva výstupy krajního obvodu předchozí řady. (Nestandardně napojené vstupy jsou vždy ohodnoceny nulou). Vstupy i–té řady základních obvodů odpovídají konfiguraci Turingova stroje před i–tým krokem výpočtu. Turingův stroj je po $t(n)$ krocích výpočtu v koncové konfiguraci, právě když nejlevější bit stavového vstupu nejlevějšího základního obvodu poslední řady je roven jedné. □

Důsledek 15.1.1 *Pokud deterministický TS přijímá jazyk $L \subseteq \{0,1\}^*$ v čase $t(n)$, pak $c_L(n) \in O(t^3(n))$.*

Důkaz: Deterministický Turingův stroj pracující v čase $t(n)$ můžeme simulovat deterministickým jednopáskovým Turingovým strojem v čase $O(t^2(n))$ a prostoru $O(t(n))$. □

Cvičení 15.2 Zamyslete se nad tím, zda by nešlo obvod zmenšit. Pokuste se vynechat podobvody, které daleko od hlavy pouze kopírují obsah pásky. Pokuste se tak jako při dvoupáskové simulaci posouvat pásky místo hlav jednotlivých pásek — tím se také ušetří nárůst kvůli redukci na jednopáskový stroj.

Věta 15.2 *Nechť L je jazyk nad abecedou $\{0,1\}$. Pak $c_L(n) \in n^{O(1)} \Leftrightarrow L \in \boldsymbol{P}/poly$.*

Důkaz: Nechť $L \in \boldsymbol{P}/poly$. Existuje tedy $B \in \boldsymbol{P}$ a $f \in poly$ tak, že $L = \{x \mid \langle x, f(|x|)\rangle \in B\}$. Nechť TS M přijímá B v čase $p(n)$. Podle důsledku 15.1.1 existuje Booleovský obvod velikosti $O(p^3(|\langle x, f(|x|)\rangle|))$ vyhodnocující $B \cap \{0,1\}^{|\langle x,f(|x|)\rangle|}$. Vytvořit ze vstupu x vstup $\langle x, f(|x|)\rangle$ je pro $f \in poly$ možno obvodem polynomiální velikosti. Složením těchto obvodů získáme polynomiálně velký obvod pro $L \cap \{0,1\}^n$.

Má-li naopak L polynomiálně velké obvody, pak stačí za $f(n)$ volit popis některého nejmenšího obvodu pro vstupy velikosti n. Vyhodnocení Booleovského obvodu je totiž problém z $\boldsymbol{P}$. □

Poznámka 15.4 Jinou charakterizací třídy $\boldsymbol{P}/poly$, kterou uvádíme bez důkazu je $\boldsymbol{P}/poly = \bigcup_S$ řídká $\boldsymbol{P}(S)$.

Poznámka 15.5 Bez důkazu uvádíme také charakterizaci třídy $\boldsymbol{P}/log$ pomocí posloupnosti booleovských obvodů. Jazyk L patří do $\boldsymbol{P}/log$ pokud existuje posloupnost $BO(i)$ booleovských obvodů, kde $L \cap \{0,1\}^n$ je rozpoznáván některým z prvních $n^{O(1)}$ obvodů. Navíc musí být i–tý obvod možno spočítat v polynomiálním čase vůči i (a protože $i^{O(1)} = (2^{|i|})^{O(1)} = 2^{O(|i|)}$, dostáváme $BO \in \boldsymbol{DExtF}$).

Rádcovská funkce poskytuje index vhodného obvodu.

Na závěr sekce použijeme „početní diagonální metodu“ k důkazu existence jazyka v $\boldsymbol{ExpSpace} \setminus \boldsymbol{P}/poly$.

Věta 15.3 *Existuje jazyk L nad abecedou $\{0,1\}$ patřící do $\boldsymbol{ExpSpace} \setminus \boldsymbol{P}/poly$.*

Důkaz: Pro dané n zkonstruujeme jazyk $L_n = L \cap \{0,1\}^n$. Jazyk L pak definujeme jakožto sjednocení takovýchto L_n.

Očíslujeme všechna slova délky n. Označíme je $x_1, \ldots, x_{2^n}$. Definujeme postupně jazyky $\emptyset = A_0 \subseteq A_1 \subseteq A_2 \subseteq \subseteq \cdots \subseteq A_{2^n} = L_n$. Vždy postupujeme tak, aby $A_k \setminus A_{k-1} \subseteq \subseteq \{x_k\}$. Slovo x_k přidáme do A_k, právě když méně než polovina z obvodů velikosti nejvýš $n^{\log n}$ přijímajících $x_1, \ldots, x_{k-1}$ ve shodě s A_{k-1} přijímá x_k.

Obvodů velikosti $n^{\log n}$ je nejvýš $2^{O(n^{\log n})}$. Nejvýš polovina z nich přijímá x_1 ve shodě s A_1. Nejvýš polovina z nich přijímá i x_2 ve shodě s A_2 ...Nejvýš 2^k-tina obvodů přijímá $x_1, \ldots, x_k$ ve shodě s A_k. Obvodů přijímajících $L_n = A_{2^n}$ je nejvýš $\frac{2^{O(n^{\log n})}}{2^{2^n}} = o(1)$. Pro dostatečně velké n neexistuje obvod velikosti nejvýš $n^{\log n}$ přijímající L_n. Libovolný polynom je od určitého n menší než $n^{\log n}$, proto neexistují polynomiálně velké obvody pro L.

K ukončení důkazu potřebujeme ukázat, že $L \in \in \boldsymbol{ExpSpace}$. K rozhodnutí, zda $x_k \in L$, potřebujeme udržovat množiny A_i postupně pro $i \leq k$. K tomu stačí prostor velikosti $O(k) = O(2^n)$. K simulaci jednotlivých obvodů velikosti $n^{\log n}$ potřebujeme prostor $n^{\log n}$ na jednotlivé obvody. Navíc potřebujeme dvě počítadla (dosud vyhovující obvody, obvody vyhovující i pro x_i). Tato počítadla jsou do čísel velikosti $2^{O(n^{\log n})}$, stačí nám na ně prostor $O(n^{\log n})$. Celkem potřebujeme prostor $O(n^{\log n})$. □

15.3 Hloubka booleovských obvodů

Lemma 15.4 *Spočítat skalární součin $(\sum_{i=1}^n x_i y_i)$ dvou vektorů délky n nad množinou $\{0,1\}$ je možno obvodem hloubky $1 + \lceil \log n \rceil$.*

Důkaz: Součiny $x_i y_i$ spočítáme $\wedge$ hradly, tato hradla pospojujeme binárním stromem s hradly $\vee$. Kořen stromu je hradlo vydávající požadovaný výsledek. □

Lemma 15.5 *Spočítat součin dvou matic rozměru $n \times n$ nad množinou $\{0,1\}$ je možno obvodem hloubky $1 + \lceil \log n \rceil$.*

Důkaz: Každý z n^2 výstupů je skalární součin dvou vektorů délky n. Ty můžeme spočítat (paralelně) podle předchozího lemmatu. □

Lemma 15.6 *Spočítat tranzitivní (reflexivně tranzitivní) uzávěr matice M rozměru $n \times n$ (nad množinou $\{0,1\}$) je možno obvodem hloubky $O(\log^2 n)$.*

Důkaz: V úvodu do složitosti jsme zjistili, že reflexivně tranzitivní uzávěr matice M můžeme spočítat podle vzorce $M^* = (M + E)^k$, kde $k \geq n$ a E je jednotková matice. Tranzitivní uzávěr můžeme spočítat podle vzorce $M^+ = = M \times M^*$. Booleovský obvod vytvoříme spojením za sebe $\lceil \log n \rceil$ obvodů umocňujících matici na druhou. Podle

předchozího lemmatu mají tyto obvody hloubku $O(\log n)$. Na první úrovni místo vstupů M_i^i použijeme vstup 1, takže první umocňování používá matici $M + E$. Výsledek je $M^{2^{\lceil \log n \rceil}} = M^*$. Vynásobení výsledku maticí M přidá hloubku $O(\log n)$. □

Věta 15.7 *Nechť $L \subseteq \{0,1\}^*$, nechť nedeterministický Turingův stroj M přijímá L v prostoru $s(n) \geq \log n$. Pak $d_L(n) \in O(s^2(n))$.*

Důkaz: Přetvoříme nejprve stroj M na stroj M', přijímající tentýž jazyk se stejnými prostorovými nároky, který má jednoznačnou přijímací konfiguraci (přepíše obsah jediné pracovní pásky nulama, vrátí se na začátek a přejde do koncového stavu q_f).

Stroj M' daný vstup x přijme, právě když je koncová konfigurace dosažitelná z počáteční konfigurace. Jinými slovy, M' přijme vstup x, jestliže v reflexivně tranzitivním uzávěru matice incidence grafu konfigurací stroje M' pro vstup x je jednička v řádku odpovídajícímu počáteční konfiguraci a sloupci odpovídajícímu koncové konfiguraci.

Počet konfigurací stroje M' je pro daný vstup délky n roven $n \cdot 2^{O(s(n))} = 2^{O(s(n))}$. Proto spočítat tranzitivní uzávěr takto velké matice umíme obvodem hloubky $O(\log^2(2^{O(s(n))})) = O(s^2(n))$.

Zbývá si uvědomit, jak hluboký obvod potřebujeme k vygenerování matice sousednosti grafu konfigurací odpovídajících vstupu x. Avšak to, zda konfigurace β může při výpočtu ihned následovat po konfiguraci α, nemůže záviset na jiných bitech vstupu, než na bitu x_i, na němž je hlava v konfiguraci α. V matici incidence je proto všude v řádku α buď 0, 1, x_i nebo $\neg x_i$. K výpočtu matice incidence nám tedy stačí obvod hloubky 1. □

15.4 Paralelismus

Tato skripta nejsou věnována paralelní složitosti. Z této oblasti zde uvádíme pouze paralelní tezi a „definici“ tříd $\boldsymbol{NC}^i$ a třídy $\boldsymbol{NC}$.

Paralelní teze tvrdí, že je polynomiální závislost mezi sekvenčním prostorem a paralelním časem.

Tato teze platí, pokud za paralelní model výpočtu vezmeme booleovské obvody.

Definice paralelní třídy nemůže být příliš korektní pokud nedefinujeme paralelní výpočetní model. V takovém modelu je potřeba definovat způsob komunikace mezi jednotlivými „procesory“. To může znamenat definici způsobu sdílení společných dat (pokud taková data existují), případně definici topologie komunikační sítě. Také musí být definován způsob zadávání vstupu.

Třídy $\boldsymbol{NC}^i$ jsou definovány jako třídy jazyků přijímaných paralelně pomocí polynomiálně mnoha procesorů v čase $O(\log^i n)$, kde n je velikost vstupu. Bližší podrobnosti výpočetního modelu bohužel neznám.

Třída $\boldsymbol{NC}$ je sjednocením všech tříd $\boldsymbol{NC}^i$.

16 P-m-log-úplnost (28. května 1998)

Dosud jsme se zabývali třídami složitosti, kde polynomiální předvýpočet neměl vliv na příslušnost do třídy. Pokud se ale chceme hlouběji zabývat strukturou třídy $\boldsymbol{P}$, potřebujeme „méně náročné“ transformace. Vhodnou transformací je transformace s logaritmickým pracovním prostorem. Budeme ji zkráceně nazývat $m - \log$ převeditelnost.

Na rozdíl od skládání časových polynomiálních transformací zde není triviálně vidět, proč by měla být složená transformace stále ještě v logaritmickém meziprostoru.

Věta 16.1 *(Tranzitivita převoditelnosti) Je-li problém A $m - \log$ převeditelný na problém B a problém B $m - \log$ převeditelný na problém C, pak je problém A $m - \log$ převeditelný na problém C.*

Důkaz: Nechť stroj M_1 transformuje zadání problému A na zadání problému B v logaritmickém prostoru. Obdobně nechť stroj M_2 transformuje zadání problému B na zadání problému C v logaritmickém prostoru. Transformace problému A na problém C bude složením transformací strojů M_1 a M_2.

Jedinou komplikací je, že nemůžeme vygenerovat zadání problému B, protože bychom tím popsali příliš mnoho pracovního prostoru.

Řešením je simulovat stroj M_2 s tím, že si místo celé „vstupní“ pásky pamatujeme pouze „pozici vstupní hlavy“ (číslo) a „čtený symbol“ na pásce pod hlavou.

Kdykoli potřebujeme posunout hlavu stroje M_2 na „vstupní“ pásce, pozměníme „pozici vstupní hlavy“ a spustíme od začátku podvýpočet stroje M_1. V tomto výpočtu nezapisujeme na výstupní pásku, ale pouze evidujeme „pozici výstupní hlavy“. Je-li „pozice výstupní hlavy“ stroje M_1 shodná s „pozicí vstupní hlavy“ stroje M_2, pak evidujeme symbol zapisovaný na výstupní pásku. Na konci simulace stroje M_1 použijeme symbol naposledy zapsaný na danou pozici výstupní pásky.

Pracovní prostor stroje M_1 je podle předpokladu nejvýš $O(\log n)$ pro vstup velikosti n. Počet konfigurací stroje M_1 je tedy nejvýš $n \cdot 2^{O(\log n)} = n^{O(1)}$, tedy výsledné slovo (jazyka B) má délku nejvýš $n^{O(1)}$. Na „pozice“ potřebujeme tedy prostor $\log n^{O(1)} = O(\log n)$. Pracovní prostor stroje M_2 je podle předpokladu $O(\log m)$, kde m je velikost vstupu, v našem případě tedy $O(\log n^{O(1)}) = O(\log n)$. Celkový pracovní prostor je součtem těchto pracovních prostorů, tedy $O(\log n)$. □

Cvičení 16.1 Mnohé z transformací zmíněných v přednášce Úvod do složitosti a $\boldsymbol{NP}$-úplnosti bylo možno provést v logaritmickém prostoru. Které?

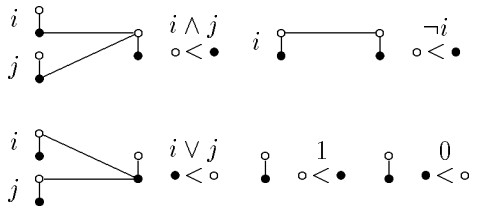
16.1 Příklady P-m-log úplných problémů

Věta 16.2 *(**CVP**) Vyhodnocení Booleovského obvodu je $P - m - \log$ úplný problém.*

Důkaz: Konstrukci z minulé kapitoly je možno provést s logaritmickým pracovním prostorem. (Musíme si pamatovat pouze řádek a sloupec.) □

Věta 16.3 *Zjistit, zda daný vrchol patří do lexikograficky minimální množiny mezi v inkluzi maximálními nezávislými množinami grafu, je $P - m - \log$ úplný problém.*

Důkaz: Ukážeme logaritmickou transformaci z problému **CVP**: Označme M lexikograficky minimální množinu mezi v inkluzi maximálními nezávislými množinami výsledného grafu. Nechť daný Booleovský obvod má n hradel. Vytvoříme graf s n dvojicemi vrcholů tak, že jak M tak každá v inkluzi maximální nezávislá množina bude obsahovat právě jeden vrchol z každé dvojice.



Obr. 8: Převod **CVP** na lexmin z maximálních v inkluzi nezávislých množin

Jeden vrchol každé dvojice budeme značit černě, druhý vrhcol budeme značit bíle. Pravdivému ohodnocení k–tého hradla bude odpovídat situace, kdy do M patří bílý vrchol k–té dvojice. Vrcholy k–té dvojice budou očíslovány čísly $2k-1, 2k$. To, zda větší číslo dostane bílý nebo černý vrchol, závisí na druhu k–tého hradla. „Simulace“ jednotlivých hradel je zobrazena na obrázku 8.

K provedení transformace potřebujeme počítadlo na zaznamenání k. Navíc vždy, když navazujeme na výstup i–tého hradla, musíme zjistit druh tohoto hradla, abychom věděli, které číslo dostal černý a které bílý vrchol. K tomu nám stačí prostor $O(\log n)$. □

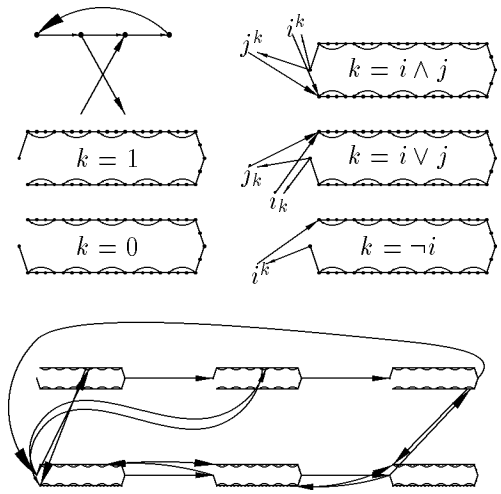
Věta 16.4 *Zjistit, zda daný vrchol patří do lexikograficky minimální cesty mezi v inkluzi maximálními cestami v orientovaném grafu je $P - m - \log$ úplný problém.*

Důkaz: Opět ukážeme logaritmickou transformaci z problému **CVP**. Označme C lexikograficky minimální cestu mezi v inkluzi maximálními cestami výsledného grafu. Nechť daný Booleovský obvod má n hradel. Vytvoříme graf na $O(n^2)$ vrcholech.

Tento graf vznikne jako orientovaná cesta, na níž bude za každé hradlo „rozdvojení“. Pravdivému ohodnocení k–tého hradla bude odpovídat situace, kdy C prochází celou horní cestou v k–tém rozdvojení.

Vrcholy cest k–tého rozdvojení jsou očíslovány čísly z intervalu $\langle k(8n+4), (k+1)(8n+4)\rangle$. „Simulace“ jednotlivých hradel je zobrazena na obrázku 9.

K provedení transformace potřebujeme počítadlo na zaznamenání k, dále potřebujeme počítat čísla vrcholů výstupu i–tého hradla. K tomu nám stačí prostor $O(\log n)$. □



Symbol i^j znamená j-tý „vstup“ horní cesty i-té rozdvojky. Obdobně i_j značí j-tý „vstup“ dolní cesty i-té rozdvojky. Graf v dolní části odpovídá obvodu $(4, 1, 2, \wedge)$, $(5, 4, -, \neg)$, $(6, 3, 5, \vee)$ pro vstup $(0, 1, 1)$.

Obr. 9: Převod **CVP** na lexmin z v inkluzi maximálních cest v orientovaném grafu

Cvičení 16.2 Zjisit, zda daný vrchol patří do lexikograficky minimální cesty mezi v inkluzi maximálními cestami v neorientovaném grafu, je $P - m - \log$ úplný problém.