Teorie vyčíslitelnosti

Petr Sosík

Slezská univerzita v Opavě
Filozoficko-přírodovědecká fakulta

Obsah

Předmluva

Tento text slouží pro základní orientaci v kursu teorie vyčíslitelnosti v rámci bakalářského a magisterského studia informatiky na Filozoficko-přírodovědecké fakultě Slezské univerzity v Opavě. Pro pochopení předpokládá elementární znalost teorie formálních jazyků a logiky v rozsahu běžného vysokoškolského kursu teoretické informatiky. Text je opatřen řadou příkladů ilustrujících typické postupy řešení problémů v probírané teorii. Některé z příkladů umožňují ilustrovat výsledek počítačovou simulací, např. formou programů s cykly nebo pomocí simulátoru Turingova stroje (který je ke stažení na WWW stránkách autora). Oddíly a příklady, které jdou nad rámec základního kursu, jsou označeny *.

Kapitola 1

Úvod

1.1 Abstraktní počítač a *Entscheigungsproblem*

Rozvoj počítačových technologií umožňuje zdvojnásobovat rychlost počítačů zhruba každé dva až tři roky; souběžně roste jejich paměťová a komunikační kapacita. Existuje ovšem principiální rozdíl v jejich výpočetních možnostech, když neuvažujeme omezení daná velikostí dostupné paměti? Jak charakterizovat abstraktní počítač s neomezenou kapacitou paměti, a třídu úloh, kterou může tento počítač vyřešit? Existují matematické úlohy, které ani s neomezenou pamětí na počítači vůbec nelze řešit?

Tyto otázky začaly být aktuální ve 30. letech 20. století, krátce před vznikem prvních elektronických počítačů. Jejich původní podobu představuje tzv. *Entscheidungsproblem,* formulovaný Davidem Hilbertem v roce 1928: je možno mechanicky (pomocí samočinného stroje) dokázat nebo vyvrátit každé tvrzení v dobře axiomatizované matematické teorii?[1]

Tento ryze matematický problém je však ekvivalentní řadě praktických úloh zásadní důležitosti: uvědomme si, že soudobé číslicové počítače kódují své vstupy a výstupy pomocí čísel a veškeré jejich operace jsou elementární matematické výpočty. Proto lze každý program přesně popsat celočíselnou funkcí, přiřazující vstupům požadované výstupy (ve formě např. binárních čísel). Každé tvrzení o počítačovém programu lze tedy převést na tvrzení o jednoduchých celočíselných funkcích (viz též kapitola 2.3).

Pokud by byl *Entscheidungsproblem* mechanicky řešitelný, bylo by například strojově možno:

- pro každý počítačový program rozhodnout, zda se může „zacyklit“;
- rozhodnout, zda proměnné programu mohou někdy nabýt jistých kritických hodnot (viz programy pro řízení lékařských přístrojů, dopravních prostředků, atomových elektráren apod.);
- je-li k dispozici přesný matematický popis zadání, ověřit zda daný program toto zadání bezchybně plní (tj. odladit jej);
- bez potřeby průběžné aktualizace, již vyžadují soudobé antiviry, rozhodnout, zda se daný program (nebo jeho část) chová jako virus.

Otázku řešitelnosti *Entscheidungsproblem* definitivně zodpověděl v roce 1931 Kurt Gödel, když ukázal, že v každém axiomatizovaném systému obsahujícím alespoň celočíselnou aritmetiku existují nutně tvrzení, která jeho vlastními prostředky nelze dokázat ani vyvrátit. Odtud bezprostředně vyplývá, že pro sebedůmyslnější číslicový počítač lze najít úlohu, kterou tento počítač nedokáže vyřešit.

[1] Hilbert měl zjednodušeně řečeno na mysli teorii s konečnou bezespornou množinou axiomů a formální logikou prvního řádu.

1.2 Modely abstraktního počítače

Abychom mohli něco usuzovat o mechanickém výpočtu, musíme nejdřív vytvořit jeho co nejjednodušší univerzální matematický model (aby bylo možno snadno zjišťovat a dokazovat jeho vlastnosti). Během 20. století vznikla řada takových modelů, od nichž se očekávalo osvětlení otázek z prvního odstavce:

- λ-kalkul, matematicko-logický aparát, z něhož mj. vychází programovací jazyk Lisp, 1933, A. Church [4];
- Turingův stroj, matematický model primitivního počítače s páskovou pamětí, 1936, A.M. Turing [17];
- teorie primitivních rekurzívních funkce, popisující třídu vyčíslitelných funkcí pomocí rekurze nezávisle na použitém počítači, 1936, S. Kleene [8];
- RASP (Random Access Stored Program), RAM (Random Access Machine), jednoduché modely počítače s libovolně adresovanou pamětí, 1963, J. Shepherdson, H. Sturgis [14];
- umělé neuronové sítě, matematické modely soustav nervových buněk, dnes s četnými praktickými aplikacemi, 1943, W.S. McCulloch, W. Pitts [10];
- další matematické modely přírodních výpočetních procesů, jako DNA výpočty, membránové výpočty a podobně, 90. léta 20. století, Adleman [1];

a mnohé další. Jak tyto modely vznikaly, bylo postupně dokázáno, že přes veškerou jejich odlišnost je jejich výpočetní síla je totožná. Tj. každý z těchto mechanismů je schopen vyřešit přesně stejnou třídu úloh jako ostatní, byť se mohou lišit efektivitou řešení. Postupně začalo být zjevné, že nejde o náhodnou shodu, že zde existuje nějaká obecná hranice mechanické vypočítatelnosti, nezávislá na použitém výpočetním systému. Toto přesvědčení, známé jako Turingova-Churchova teze, poprvé zformuloval A. Church [3] v roce 1936:

> *Každý výpočetní proces, který přirozeně chápeme jako efektivně vypočitatelný, může být realizován Turingovým strojem.*

Pojem *efektivně vypočitatelný* se zde nevztahuje k rychlosti výpočtu apod., ale označuje proces, který vždy po *konečném počtu* operací najde požadovaný výsledek (pokud existuje). Teorie vyčíslitelnosti tedy zkoumá, vycházejíc dodnes z Turingovy-Churchovy teze, mechanickou řešitelnost úloh různých typů, a současně se hledá způsoby, jak překročit či obejít výpočetní limit daný T.-Ch. tezí.

Turingova-Churchova teze není matematickým tvrzením, které by mohlo být formálně dokázáno nebo vyvráceno. Vyjadřuje vlastně názor na to, které úlohy je z hlediska informatiky vhodné považovat za efektivně vypočitatelné (vyčíslitelné), a které ne. Je stále považována za velmi užitečnou a Turingův stroj je všeobecně užíván jako měřítko vyčíslitelnosti, neboť:

- takto vymezená třída efektivně řešitelných úloh je velmi přirozená a nezávislá na typu použitého výpočetního mechanismu;
- tato třída chápaná jako třída vypočitatelných funkcí má silné uzávěrové vlastnosti vzhledem k řadě operací (skládání, rekurze, inverze atd.);
- není znám žádný uzavřený výpočetní systém vymykající se Turingově-Churchově tezi, tzn. který by dokázal efektivně řešit úlohy nevypočitatelné Turingovým strojem;

– výpočetní schopnost výše uvedených modelů počítače se nemění zvyšováním jejich výkonnosti či parametrů (více paměti, rychlejší výpočet, důmyslnější operace, paralelismus, pravděpodobnostní výpočty apod.).

Přestože není Turingova-Churchova teze exaktním matematickým výrokem, může se čtenář v často i v exaktních důkazech setkat s formulacemi typu „strojová řešitelnost úlohy plyne z Turingovy-Churchovy teze.“ Takovou formulací je míněno přesvědčení dokazujícího, že ona úloha je zcela zjevně algoritmizovatelná, že on (i čtenář) by v případě potřeby snadno napsal příslušný program, že však považuje za zbytečné rozvádět věc do takových detailů.

Tento zkratkovitý postup si můžeme dovolit proto, že je řadu let dokázána výpočetní ekvivalence všech možných programovacích prostředků s Turingovým strojem, a že jsme u většiny úloh dík svým zkušenostem schopni jejich programovou řešitelnost rozpoznat. Vždy však musíme být v případě pochybností připraveni sestavit příslušný algoritmus.

1.3 Churchova-Turingova teze v širším kontextu

Jedním z klíčových problémů teorie vyčíslitelnosti je vztah výše zmíněných matematických modelů počítače (pracujících vesměs s konečnou množinou symbolů) k fyzikální realitě. Budeme-li považovat za efektivně vyčíslitelný každý „deterministický“ fyzikální proces, je otázkou, zdali existuje fyzikální obdoba Turingova stroje. Ze zákonů kvantové fyziky skutečně plyne teoretická možnost existence univerzálního *kvantového Turingova stroje*, který je výpočetně mocnější než běžný Turingův stroj a může efektivně simulovat libovolný fyzikální proces. Pokud by byl realizován, mohl by např. provést libovolný fyzikální experiment.

Pokud by se i lidská mysl chovala mechanicky, platil by zdánlivě limit strojové vyčíslitelnosti daný Turingovým strojem i pro všechno lidské poznání. Existují důvody pro i proti tomuto tvrzení (viz univerzální kvantový Turingův stroj – může být mysl vysvětlena a popsána fyzikálními zákony?), o nichž se intenzívně diskutuje zejména mezi experty na umělou inteligenci.

Při srovnání Turingova stroje a lidského mozku (a nebo dokonce běžného soudobého počítače připojeného k Internetu) však vyniknou přinejmenším následující rozdíly:

Kontinuita – Turingův stroj nemá možnost přenosu informace mezi jednotlivými běhy při řešení úloh. Veškerá informace získaná při řešení jedné úlohy je po jejím ukončení zcela zapomenuta. Naproti tomu např. osobní počítač může na svých médiích akumulovat poznatky získané při řešení úloh a v budoucnu je využít.

Interaktivita – Turingův stroj nemůže nijak interagovat se svým prostředím, obdrží jednorázově zadání úlohy na začátku činnosti. Nemá možnost klást doplňující dotazy, „radit se“ s jinými stroji a podobně. Přitom právě faktor *nepředvídatelnosti* v prostředí bohatém na interakce, jako je Internet, se ukazuje jako možná významná posila výpočetní síly.

Adaptabilita – Turingův stroj se nemůže nijak přestavět, zjistí-li, že není schopen vyřešit zadanou úlohu. Naproti tomu počítač může být uživatelem upgradován, mozek si vytváří nové synapse atp.

Tyto rozdíly povedeme v patrnosti, i když v dalším textu budeme uvádět, že libovolný počítač může být modelován Turingovým strojem. Mohou interaktivní systémy s

uvedenými vlastnostmi překonat výpočetní limit Turingova stroje? Jak přirozeně vymezit vlastnosti takového „silnějšího“ výpočetního modelu? Je pro ně vůbec pojem *výpočet* nebo *mechanický postup* adekvátní? Odpovědi na tyto otázky jsou dosud otevřené. Zajímavé úvahy a odkazy na toto téma lze nalézt např. v [7] nebo [18].

Kapitola 2

Reprezentace strojově vyčíslitelných funkcí

2.1 Turingův stroj

Ze výše uvedených modelů mechanického výpočtu je pro teoretické studie nejčastěji užíván Turingův stroj, neboť

- jde o intuitivně pochopitelný model skutečného, byť primitivního počítače s páskovou pamětí,
- je snadno programovatelný pro vykonání zadané úlohy (na rozdíl od např. ryze matematických modelů),
- svou jednoduchostí dovoluje podávat důkazy v relativně kompaktní formě (např. model RAM je podstatně složitější).

Řada důležitých tvrzení o algoritmech a počítačích byla zjištěna a dokázána právě pomocí tohoto modelu. Každý dnes existující počítač totiž lze Turingovým strojem simulovat (s výhradami uvedenými v kapitole 1.3), i když často velmi neefektivním způsobem.

Definice 2.1.1 *Turingův stroj (TS) je šestice* $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, *kde:*

Q *je konečná množina stavů,*
Γ *je konečná pásková abeceda obsahující mj. symbol mezera* $\sqcup$,
$\Sigma \subseteq \Gamma - \{\sqcup\}$ *je vstupní abeceda,*
$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\leftarrow, \downarrow, \rightarrow\}$, *je přechodové zobrazení, kde* $DOM(\delta) \subseteq Q \times \Gamma$,
$q_0 \in Q$ *je počáteční stav,*
$F \subseteq Q$ *je množina koncových stavů.*

Symbol $\sqcup$ se též označuje jako prázdný symbol. Při popisech v dalším textu budeme používat symboly Q, Σ, Γ, δ, q_0, F ve výše popsaném významu, nebude-li řečeno jinak. Zápis DOM(δ) označuje definiční obor zobrazení δ.

Turingův stroj disponuje obousměrně nekonečnou páskou, rozdělenou na políčka, obsahující po jednom symbolu z Γ. V každém kroku stroj pomocí své tzv. čtecí/zapisovací hlavy (model vznikal v době, kdy byly rozšířeny děrné pásky) přečte jeden symbol z pásky, podle jeho obsahu a podle svého vnitřního stavu (z množiny Q) jej přepíše nějakým symbolem z Γ (ne nutně jiným), změní svůj vnitřní stav a přesune hlavu na pásce o jedno políčko vlevo, vpravo nebo zůstane stát. Existují varianty definice 2.1.1, např. stroj s jednosměrně nekonečnou páskou, zákaz zápisu mezery $\sqcup$ na pásku, jen jeden koncový stav, speciální vstupní a výstupní páska apod. Lze však ukázat, že těmito změnami se nijak nezmění výpočetní síla stroje.

Abychom mohli činnost stroje přesně definovat, je třeba nejdřív popsat, v jaké situaci se stroj nachází před provedením kroku, a jaké změny tento krok způsobí. Úplná informace o situaci stroje v jistém momentě se nazývá *konfigurace*.

Definice 2.1.2 Konfigurace *Turingova stroje je trojice* (q, w, w'), *kde* $q \in Q$, $w, w' \in \Gamma^*$.

Dohoda: ve slovech w, resp. w' nebudeme zapisovat mezery zleva, resp. zprava, jichž je nekonečně mnoho.

Slovo ww' představuje obsah pásky, pozice hlavy stroje je na prvním symbolu slova w'. Jelikož vždy pouze konečné množství symbolů na pásce bude nemezerových, tedy zápis slova ww' bude konečný. Popis konkrétního Turingova stroje (podle definice) a jeho aktuální konfigurace přesně určuje, jak bude jeho činnost dále probíhat a jakými dalšími konfiguracemi stroj projde.

Definice 2.1.3 *Buď* (p, xa, by) *konfigurace stroje* M *pro nějaká* $a, b \in \Gamma$, $x, y \in \Gamma^*$, $p \in Q$. *Krok* $\vdash_M$ *stroje* M *je přechod do jiné konfigurace podle některého z těchto předpisů:*

- $(p, xa, by) \vdash_M (q, xa, cy)$, *pokud* $\delta(p, b) = (q, c, \downarrow)$;
- $(p, xa, by) \vdash_M (q, x, acy)$, *pokud* $\delta(p, b) = (q, c, \leftarrow)$;
- $(p, xa, by) \vdash_M (q, xac, y)$, *pokud* $\delta(p, b) = (q, c, \rightarrow)$.

Krok je tedy chápán jako binární relace, obsahující dvojice takových konfigurací, mezi nimiž stroj přejde jednou svou elementární operací.

Tranzitivní, resp. reflexivní a tranzitivní uzávěr relace $\vdash_M$ označujeme $\vdash_M^+$, resp. $\vdash_M^*$. Zápis $x \vdash_M^+ y$ neznamená nic jiného, než že stroj M přepíše slovo x na pásce na slovo y pomocí libovolného nenulového počtu kroků. Zápis $x \vdash_M^* y$ pak navíc zahrnuje i možnost, že nebyl vykonán žádný krok a $x = y$.

Na začátku činnosti stroje je na pásce konečně dlouhé tzv. vstupní slovo $x \in \Sigma^*$, představující zadání nějakého problému dodané zvnějšku. Všechna ostatní políčka pásky obsahují mezery. Stroj během činnosti toto slovo přetváří na nějaké jiné slovo $w \in \Gamma^*$, které může představovat řešení problému. Činnost TS může vyústit jedním z následujících způsobů:

1. Stroj přejde do koncového stavu. V tom případě se zastaví a tzv. *akceptuje (přijme)* vstupní slovo, které bylo na původně na pásce.

2. Stroj se dostane do konfigurace (q, w, aw'), kde $a \in \Gamma$ a platí $(q, a) \notin \mathrm{DOM}(\delta)$. Tedy stroj pro kombinaci stavu, v němž se nachází, a symbolu, který čte, nemá definovánu žádnou činnost. Pak se rovněž zastaví, ale *neakceptuje* vstup.

3. Nenastane žádná z předchozích variant a stroj se vůbec nezastaví, tj. donekonečna koná kroky. V tom případě opět *neakceptuje* vstup.

Definice 2.1.4 *(i) Jazyk* přijímaný *Turingůvým strojem* M *je množina*

$$L(M) = \{x \in \Sigma^* | (q_0, \epsilon, x) \vdash_M^* (q_F, w, w')\},$$

kde $q_F \in F$ *a* w, w' *jsou libovolná slova z* Γ^*.

(ii) Množina $A \subseteq \mathbb{N}^n$ je přijímána *Turingovým strojem M se vstupní abecedou $\{0,1\}$, pokud platí*

$$(x_1, \ldots, x_n) \in A \qquad \Longleftrightarrow \qquad 1^{x_1+1}0 \ldots 01^{x_n+1} \in L(M).$$

Příklad 2.1.5 *Uvažujme stroj $M = (\{q_0, q_1, q_F\}, \{a, b\}, \{a, b, \sqcup\}, \delta, q_0, \{q_F\})$, kde $\delta(q_0, a) = (q_1, a, \rightarrow)$, $\delta(q_1, \sqcup) = (q_1, \sqcup, \rightarrow)$, $\delta(q_1, a) = (q_1, a, \rightarrow)$, $\delta(q_1, b) = (q_F, b, \downarrow)$. Tento stroj se*

- *zastaví a akceptuje pro slova začínající a a obsahující b;*
- *zastaví a neakceptuje pro slova nezačínající a;*
- *nezastaví pro ostatní slova (začínající a a neobsahující b).*

Jazyk přijímaný tímto strojem je množina $L(M) = \{a\}^+\{b\}\{a, b\}^*$.

Pokud se stroj M se vstupem x zastaví (jedním ze dvou výše uvedených způsobů, ať už v koncovém či jiném stavu), píšeme $M(x) = \searrow$. V opačném případě píšeme $M(x) = \nearrow$. Vzhledem k tomu, že stroj v koncovém stavu nekoná žádné kroky, budeme dále u všech strojů předpokládat, že $\mathrm{DOM}(\delta) \subseteq (Q - F) \times \Gamma$.

Poznámka 2.1.6 Alternativní definice, kterou použijeme v některých příkladech, počítá s jediným koncovým stavem nazvaným *ACCEPT*, a se speciálním stavem $REJECT \in Q$, přičemž $\mathrm{DOM}(\delta) = (Q - \{ACCEPT, REJECT\}) \times \Gamma$. Stroj se tedy vždy zastaví v jednom ze stavů *ACCEPT*, *REJECT*, přičemž ve stavu *REJECT* neakceptuje vstupní slovo.

Libovolný TS z definice 2.1.1 lze takto upravit, aniž by se změnila jeho funkce. Jednoduše přidáme do Q nové stavy *ACCEPT*, *REJECT* a upravíme δ tak, že

- pro všechny dvojice (q, a), $q \in F$, $a \in \Gamma$ definujeme $\delta(q, a) = (ACCEPT, a, \downarrow)$;
- pro všechny dvojice $(q, a) \notin \mathrm{DOM}(\delta)$, $q \in (Q - F)$, $a \in \Gamma$ definujeme $\delta(q, a) = (REJECT, a, \downarrow)$;
- položíme $F = \{ACCEPT\}$.

Cvičení 2.1.7 *Sestrojte TS rozpoznávající jazyk všech dobře utvořených výrazů z lomených závorek, tedy správně $\langle\rangle$, $\langle\langle\rangle\rangle\langle\rangle$, nesprávně $\rangle\langle$, $\langle\rangle\rangle$ atd.*

2.1.1 Vyčíslování funkcí Turingovým strojem

Výstupem Turingova stroje budeme rozumět obsah pásky v okamžiku zastavení.

Definice 2.1.8 *Platí-li pro stroj M : $(q_0, \epsilon, x) \vdash_M^* (q, y, y')$ pro $q \in Q$, $x \in \Sigma^*$, $y, y' \in \Gamma^*$ a stroj M se zastaví v konfiguraci (q, y, y'), píšeme $M(x) = yy'$.*

Použitý znak ϵ představuje prázdné slovo, tedy slovo délky 0, neobsahující žádný symbol. To je nutno odlišovat od zápisu $\sqcup$, který představuje mezeru, anebo slovo délky 1 sestávající z jedné mezery.

Nehrozí-li nebezpečí nedorozumění, budeme souběžně používat zápisy $M(x) = yy'$ a $M(x) = \searrow$. Přitom první znamená, že stroj se vstupem x vyprodukoval výstup yy', a druhý pouze fakt, že se stroj se vstupem x zastavil.

Stroj tedy podle definice 2.1.8 přetvoří vstupní slovo na pásce na odpovídající výstup yy'. Proto jeho činnost lze chápat i tak, že vypočítává funkci f nad řetězci, která přiřazuje vstupnímu slovu x odpovídající výstupní slovo ww'. Jestliže vstupní slova představují zakódovaná čísla, může stroj počítat i nějakou číselnou funkci. Pokud se stroj pro nějaké vstupní slovo x nezastaví, hodnota $f(x)$ není definována.

Definice 2.1.9 *Výpočet funkcí Turingovým strojem.*

1. *(Částečná) funkce nad řetězci* $f : \Sigma^* \longrightarrow \Gamma^*$ *je vyčíslována strojem* M, *platí-li pro všechna* $x \in \Sigma^*$:

$$M(x) = \begin{cases} f(x) & \text{je-li } f(x) \text{ definována;} \\ \nearrow & \text{jinak.} \end{cases}$$

2. *(Částečná) funkce* $f : \mathbb{N}^t \longrightarrow \mathbb{N}^s$ *je vyčíslována strojem* M, *platí-li pro všechna* $(x_1, \ldots, x_t) \in \mathbb{N}^t$:

$$M(1^{x_1+1}01^{x_2+1}0\ldots01^{x_t+1}) = \begin{cases} 1^{y_1+1}01^{y_2+1}0\ldots01^{y_s+1}, & \text{pokud } f(x_1, \ldots, x_t) = (y_1, \ldots, y_s); \\ \nearrow & \text{je-li } f(x_1, \ldots, x_t) \text{ nedefinována.} \end{cases}$$

Pokud nehrozí nedorozumění, budeme místo $M(1^{x_1+1}0\ldots01^{x_t+1}) = 1^{y_1+1}0\ldots01^{y_s+1}$ zapisovat zkráceně $M(x_1, \ldots, x_t) = (y_1, \ldots, y_s)$.

Je-li funkce f vyčíslována nějakým Turingovým strojem M, nazývá se f *(částečně) vyčíslitelná* funkce. Slovo „částečně“ vynecháváme, pokud f je totální funkce, tedy když $\mathrm{DOM}(f) = \Sigma^*$, resp. $\mathrm{DOM}(f) = \mathbb{N}^t$.

Příklad 2.1.10 *Následující TS vyčísluje funkci* $f(x) = x + 1$:

$$M = (\{q_0, q_F\}, \{1\}, \{1, \sqcup\}, \delta, q_0, \{q_F\}),$$

kde $\delta(q_0, 1) = (q_0, 1, \rightarrow)$, $\delta(q_0, \sqcup) = (q_F, 1, \rightarrow)$.

Poznámka 2.1.11 Výpočetní proces realizovatelný na Turingově stroji, který se zastaví pro každý vstup, budeme označovat pojmem *algoritmus*. Pojmem *procedura* označíme proces realizovatelný na Turingově stroji, který se pro některé vstupy nemusí zastavit. Tedy každý algoritmus je i procedura, ne každá procedura je však algoritmus.

V dalších kapitolách budeme často pracovat s tvrzením, že jeden Turingův stroj (nebo obecně procedura) *simuluje* druhý. *Simulací* stroje M strojem M' rozumíme stav, kdy pomocí jednoduchého *algoritmu* dokážeme z konfigurací $k'_0, k'_1, k'_2, \ldots$, jimiž projde M' se vstupem x, odvodit konfigurace $k_0, k_1, k_2, \ldots$, jimiž by prošel M s tímtéž vstupem, pro libovolné $x \in \Sigma^*$.

Cvičení 2.1.12 *Sestrojte TS sčítající dvě čísla, tedy vyčíslující funkci* $f(x, y) = x + y$.

2.1.2 Metody „programování“ Turingova stroje

V této kapitole ukážeme několik jednoduchých triků a konvencí, jak zjednodušit zápis pravidel a usnadnit konstrukci Turingových strojů. Níže uvedené postupy nepředstavují odchylku od definice 2.1.1, pouze ukazují na možnosti v ní obsažené.

Nejprve zavedeme následující konvenci: nechť pro jisté stavy $p, q \in Q$, posun $D \in \{\leftarrow, \downarrow, \rightarrow\}$ a množinu symbolů $\{a_1, a_2, \ldots, a_n\} \subseteq \Gamma$ obsahuje přechodové zobrazení stroje pravidla

$$\delta(p, a_1) = (q, a_1, D),\ \delta(p, a_2) = (q, a_2, D),\ \ldots,\ \delta(p, a_n) = (q, a_n, D).$$

Takovou množinu pravidel zapíšeme zkráceně jediným zápisem $\delta(p, [a_1, \ldots, a_n]) = (q, *, D)$.

Rozdělení pásky na stopy

V některých případech je užitečné si představit pásku stroje jako vícestopou, přičemž vstup (a výstup) stroje je vždy na první stopě, ostatní stopy slouží jako pomocná paměť pro zápis symbolů z Γ či jiných informací. Pro n-stopou pásku, $n \geq 1$, stačí namísto množiny Γ použít $\Gamma' = \Gamma^n$. Prvky Γ' jsou n-tice symbolů z Γ. Pro $1 \leq i \leq n$ odpovídá i-tá složka n-tice symbolu na i-té stopě.

Při popisu páskové abecedy takového stroje budeme pro zjednodušení uvádět raději výčet symbolů Γ než výčet všech možných n-tic v Γ', kterých může být mnoho. Pravidla (přechodové zobrazení) budeme zapisovat ve tvaru

$$\delta(p, b_1, \ldots, b_n) = (q, c_1, \ldots, c_n, D),$$

kde $p, q \in Q$, $b_1, \ldots, b_n, c_1, \ldots, c_n \in \Gamma$, $D \in \{\leftarrow, \downarrow, \rightarrow\}$.

Cvičení 2.1.13 *Sestrojte dvoustopý TS s páskovou abecedou $\{1, \sqcup\}$ počítající funkci $f(n) = 2n$.*

Podprogramy

Podobně jako v praktickém programování je vhodné při konstrukci Turinova stroje rozdělit úkol na jednodušší moduly. To lze snadno provést konstrukcí samostatných strojů pro jednotlivé moduly a jejich propojením pomocí počátečních a koncových stavů do výsledného stroje.

Buď $M_1 = (Q_1, \Sigma, \Gamma, \delta_1, q_{10}, \{q_{1F}\})$, $M_2 = (Q_2, \Gamma - \{\sqcup\}, \Gamma, \delta_2, q_{20}, \{q_{2F}\})$ stroje takové, že $Q_1 \cap Q_2 = \{q_{20}, q_{2F}\}$, přičemž q_{20} není na levé straně žádného pravidla z δ_1 (a q_{2F} není na levé straně žádného pravidla z δ_2, což jsme dohodli již dříve). Spojme tyto stroje do jediného $M = (Q_1 \cup Q_2, \Sigma, \Gamma, \delta_1 \cup \delta_2, q_{10}, \{q_{1F}\})$.

M po spuštění nejprve simuluje činnost M_1. Ten může vyvolat M_2 jako podprogram tím, že přejde do jeho počátečního stavu q_{20}. Nyní může konat kroky pouze stroj M_2, dokud nepřejde do svého koncového stavu, z něhož může opět navázat stroj M_1.

Cvičení 2.1.14 *Sestrojte TS s páskovou abecedou $\{1, \sqcup\}$ počítající funkci $f(n) = 2^n$, který využívá stroj z příkladu 2.1.13 jako podprogram.*

2.1.3 Nedeterministický Turingův stroj

U některých typů formálních automatů, např. u zásobníkového automatu, je nedeterministická varianta výpočetně silnější než deterministická. Řešme tuto otázku pro případ Turingovy stroje. Kladná odpověď by mj. znamenala, že nedeterministické, náhodně se rozhodující programy by měly větší výpočetní sílu než determinidstické.

Nedeterministický Turingův stroj může mít v každé konfiguraci více než jednu možnost, jaký vykonat krok. Přechodová relace (již to není zobrazení!) tedy může svazovat každou dvojici $\langle$ *stav*, *symbol* $\rangle$ s několika možnými reakcemi stroje.

Definice 2.1.15 *Nedeterministický Turingův stroj je šestice* $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, *kde* Q, Γ, Σ, q_0 *a* F *je jako v definici 2.1.1 a*

$$\delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{\leftarrow, \downarrow, \rightarrow\}$$

je přechodová relace určující chování stroje.

Krok nedeterministického TS je pak definován následovně:

$$(p, x, by) \vdash_M (q, x, cy), \text{ pokud } (p, b, q, c, \downarrow) \in \delta,$$

a obdobně pro zbývající dva směry posunu, viz definice 2.1.3. Jazyk přijímaný nedeterministickým TS je definován stejně jako u deterministického stroje. Tzn. slovo je přijato tehdy, jestliže *existuje* posloupnost kroků vedoucích do koncové konfigurace, čili *alespoň jedna* z možných sekvencí nedeterministických rozhodnutí stroje vede do koncového stavu.

Věta 2.1.16 *Buď* $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ *nedeterministický Turingův stroj. Pak existuje deterministický Turingův stroj* M' *tak, že* $L(M) = L(M')$ *pro všechna* $x \in \Sigma^*$.

Důkaz. Nechť k je velikost největší podmnožiny δ takové, v níž mají všechny pětice (p, a, q, b, D) shodný stav p a symbol a, tedy k je maximální počet rozhodnutí stroje v libovolné situaci. Uvažujme třístopý TS M', který začíná s originálním vstupem stroje M na 1. stopě. Na 2. stopě generuje postupně pro $m = 0, 1, 2, \ldots$ všechny posloupnosti číslic $i_1, i_2, \ldots, i_m$, kde $i_j \in \langle 1, k \rangle$ pro všechna j, $1 \leq j \leq m$.

Pro každou takovou posloupnost délky m na 2. pásce stroj přepíše vstup z 1. na 3. stopu a simuluje na 3. stopě m kroků stroje M. Přitom pracuje s posloupností rozhodnutí z 2. stopy, tedy v k-tém kroku vybere i_k-tou variantu kroku stroje M. (Pokud má stroj M v tomto kroku méně než i_k možností rozhodnutí, stroj M' vybere první z nich.) Takto projde postupně všechny možnosti chování stroje M. Pokud alespoň jedna z nich vede do koncového stavu, M' se zastaví a akceptuje. Tedy pro každé vstupní slovo x platí $x \in L(M) \iff x \in L(M')$, což dává $L(M) = L(M')$. □

2.2 While programy

V této kapitole ukážeme, že k dosažení univerzální výpočetní síly (tj. síly Turingova stroje) v libovolném programovacím jazyce stačí jednoduché přiřazovací a aritmetické příkazy, podmíněný příkaz a příkaz cyklu. Tento výsledek nás navíc ubezpečuje, že každý program obsahující skoky typu **goto** lze nahradit funkčně ekvivalentním strukturovaným programem bez skoků.

Použijeme primitivní programovací jazyk pracující pouze s číselnými proměnnými, můžeme jím ovšem reprezentovat i programy, jejichž vstupem nebo výstupem jsou řetězce, obrázky apod. Stačí si uvědomit, že v číslicových počítačích jsou veškerá data reprezentována čísly, vstupní rozhraní počítače konvertuje veškeré vstupy na čísla, a obrácená konverze probíhá na výstupu.

Definice 2.2.1 **While** programem *nazveme program s množinou proměnných* $\mathbf{Var} = \{x, y, \ldots\}$, *jež nabývají hodnot z* $\mathbb{N}$, *vytvořený induktivně z následujících konstrukcí:*

(i)	*přiřazovací příkazy*	$x := 0 \quad x := y + 1 \quad x := y$
(ii)	*složený příkaz*	**begin** P; Q $\ldots$ **end**
(iii)	*podmíněný příkaz*	**if** $x < y$ **then** P **else** Q
(iv)	*cyklus for*	**for** y **do** P
(v)	*cyklus while*	**while** $x < y$ **do** P

Výpočetní síla **while** programů se nezmění, nahradíme-li relaci $<$ kteroukoli z relací $>, \leq, \geq, =, \neq$. Programy vytvořené jen pomocí konstrukcí (i) až (iv) se nazývají **for** programy.

Při vykonávání cyklu **for** y **do** P je nejprve určena aktuální hodnota proměnné y a pak je program P vykonán y-krát. Provádění cyklu nemění obsah proměnné y, ledaže je změněna přiřazovacím příkazem uvnitř P. Změna hodnoty y uvnitř P však nemá vliv na počet opakování cyklu. Cyklus **for** se tedy vždy po konečném počtu opakování ukončí.

Při vykonávání cyklu **while** $x < y$ **do** P je nejprve vyhodnocena podmínka $x < y$. Není-li splněna, tělo cyklu P není vykonáno a program pokračuje příkazem následujícím za cyklem. Je-li podmínky splněna, je vykonán program P a vše se opakuje s novými hodnotami x, y (které mohly být eventuelně změněny uvnitř P).

Poznámka 2.2.2 Ve **while** programech není nutno užívat cyklus **for** y **do** P, neboť jej lze nahradit cyklem **while** následovně:

$$z := 0; \quad w := y; \quad \textbf{while } z < w \textbf{ do begin } \quad P; \quad z := z + 1 \quad \textbf{end}$$

kde z a w jsou pomocné proměnné nepoužité v P. Důvody zavedení cyklu **for** objasňuje věta 2.3.17. Všimněme si, že program obsahující pouze cykly **for** se vždy zastaví, neboť počet opakování každého cyklu je pevně určen ještě před jeho provedením. To však nemusí platit pro cykly **while** , viz příklad 2.2.9.

Příklad 2.2.3 *Následující* **for** *program počítá funkci* $y := \begin{cases} x - 1 & \text{pro } x > 0, \\ 0 & \text{pro } x = 0. \end{cases}$

$$z := 0; \quad y := 0; \quad \textbf{for } x \textbf{ do begin } \quad y := z; \quad z := z + 1 \quad \textbf{end}$$

Příklad 2.2.4 *Následující* **for** *program počítá funkci* $z := x * y$:

$$z := 0; \quad \textbf{for } x \textbf{ do } \quad \textbf{for } y \textbf{ do } \quad z := z + 1$$

Cvičení 2.2.5 *Ukažte, že podmíněný příkaz* **if** $x < y$ **then** P **else** Q *se dá nahradit pomocí cyklu* **while** .

Cvičení 2.2.6 *Vytvořte* **for** *program počítající funkci* $n!$

Cvičení 2.2.7 *Vytvořte* **while** *program počítající funkce* 2^n *a* $\lceil \log_2 n \rceil$.

Cvičení 2.2.8 *Vytvořte* **while** *programy počítající funkce div (celočíselné dělení) a mod (zbytek po dělení).*

2.2.1 Výpočetní síla while programů

Ukážeme, že **while** programy dokáží počítat všechny částečně vyčíslitelné funkce. Odtud plyne, že v každém programovacím jazyce obsahujícím nejméně konstrukce (i), (ii), (iii) a (v) lze podle Turingovy-Churchovy teze naprogramovat libovolný efektivně vypočítatelný postup.

K důkazu je zapotřebí nejprve formálně popsat výpočty prováděné **while** programy. *Stavem prostředí* budeme nazývat zobrazení σ : **Var** $\longrightarrow \mathbb{N}$, přiřazující každé proměnné z **Var** nějaké nezáporné celé číslo. $\sigma(x)$ tedy označuje hodnotu proměnné x ve stavu prostředí σ. Množinu všech stavů prostředí označíme **Env**.

While program začne pracovat s nějakým stavem prostředí σ a během výpočtu může přiřadit proměnným jiné hodnoty, čímž změní stav prostředí. Program P lze tedy chápat jako funkci $[P]$: **Env** $\longrightarrow$ **Env**. Výsledný stav prostředí označíme $[P](\sigma)$. Pokud se P pro některý počáteční stav prostředí σ nezastaví (obsahuje nekonečný **while** cyklus), je funkce $[P]$ částečná a hodnota $[P](\sigma)$ není pro toto σ definována.

Příklad 2.2.9 *Uvažujme program* **while** $x > 0$ **do** $x := x + 1$. *Tento program se zastaví pro* $\sigma(x) = 0$, *ale nezastaví se např. pro* $\sigma(x) = 1$.

Věta 2.2.10 *Pro každou částečně vyčíslitelnou funkci* $f : \mathbb{N}^n \longrightarrow \mathbb{N}$ *existuje* **while** *program* P *s množinou proměnných* **Var** $\supseteq \{x_0, x_1, \ldots, x_n\}$ *tak, že pro libovolný stav prostředí* σ *platí:*

$f(\sigma(x_1), \ldots, \sigma(x_n))$ *je definována* $\iff$ $[P](\sigma)$ *je definována, a jsou-li obě definovány, pak*

$$[P](\sigma)(x_0) = f(\sigma(x_1), \ldots, \sigma(x_n)). \tag{2.1}$$

Důkaz. Uvažujme Turingův stroj $M = (Q, \Sigma, \Gamma, \delta, q_0, \{ACCEPT\})$ (viz poznámka 2.1.6) počítající funkci f. Uvažujme bez újmy na obecnosti $REJECT{=}ACCEPT$, tedy zastaví-li se náš stroj, pak vždy v tomto jediném stavu. Označme

$$\begin{aligned} Q &= \{q_0, q_1, \ldots, q_k\}, \quad k \geq 0, \quad \text{kde } q_k = ACCEPT = REJECT, \\ \Gamma &= \{a_0, \ldots, a_{m-1}\}, \quad m \geq 1, \quad \text{kde } a_0 = \sqcup, \\ \delta &= \{\delta_1, \ldots, \delta_{k \cdot m}\}. \end{aligned}$$

V posledním řádku symboly δ_h, $1 \leq h \leq k \cdot m$ označují prvky zobrazení δ, tedy pětice $(q_i, a_j, q_{i'}, a_{j'}, D)$, kde $q_i, q_{i'} \in Q$, $a_j, a_{j'} \in \Gamma$, $D \in \{\rightarrow, \downarrow, \rightarrow\}$.

Nechť ve **Var** existují proměnné x_q, x_L a x_R, které budou reprezentovat každou konfiguraci (q, w, w') stroje M takto:

- je-li $q = q_i$, $1 \leq i \leq n$, pak $x_q = i$;
- je-li $w = a_{i_1} a_{i_2} \ldots a_{i_k}$, $k \geq 0$, pak $x_L = i_k m^0 + i_{k-1} m^1 + \ldots + i_1 m^{k-1}$;
- je-li $w' = a_{j_1} a_{j_2} \ldots a_{j_{k'}}$, $k' \geq 0$, pak $x_R = j_1 m^0 + j_2 m^1 + \ldots + j_{k'} m^{k'-1}$.

Proměnná x_q obsahuje číslo aktuálního stavu a proměnné x_L a x_R kódují levou, resp. pravou část pásky stroje M vzhledem k poloze hlavy. Čtenář se snadno přesvědčí, že tato reprezentace je jednoznačná, a že pro indexy páskových symbolů pod hlavou stroje, resp. na políčku vlevo od hlavy platí

$$\begin{aligned} i_k &= x_L \bmod m, \\ j_1 &= x_R \bmod m. \end{aligned}$$

Možnost aplikace pravidla δ_h, $1 \leq h \leq k \cdot m$, a příslušnou změnu konfigurace M pak vypočteme následujícím programem P_h :

- Je-li $\delta_h = (q_i, a_j, q_{i'}, a_{j'}, \rightarrow)$, pak P_h má tvar

  if $x_q = i$ **then if** $(x_R \bmod m) = j$ **then begin**
  $x_q := i'$; $x_L := x_L * m + j'$; $x_R := x_R \text{ div } m$
  end

- Je-li $\delta_h = (q_i, a_j, q_{i'}, a_{j'}, \leftarrow)$, pak P_h má tvar

  if $x_q = i$ **then if** $(x_R \bmod m) = j$ **then begin**
  $x_q := i'$; $x_R := ((x_R \text{ div } m) * m + j') * m + (x_L \bmod m)$; $x_L := x_L \text{ div } m$
  end

- Je-li $\delta_h = (q_i, a_j, q_{i'}, a_{j'}, \downarrow)$, pak P_h má tvar

  if $x_q = i$ **then if** $(x_R \bmod m) = j$ **then begin**
  $x_q := i'$; $x_R := (x_R \text{ div } m) * m + j'$
  end

Díky cvičením a příkladům z této kapitoly víme, že všechny funkce v tomto popisu jsou vyčíslitelné **while** programy. Předpokládejme dále, že hlava stroje M po ukončení výpočtu ukazuje vždy na nejlevější ze symbolů 1 reprezentujících výsledek. (Není obtížné upravit stroj M tak, aby tuto podmínku splňoval). Pak jeho koncová konfigurace bude vždy $(q, \epsilon, 1^{y+1})$ pro nějaké $q \in Q$ a $y \geq 0$.

Nyní již snadno sestavíme **while** program počítající funkci f. Uvažujme pro jednoduchost $n = 1$, tedy vstupem stroje M má být řetězec 1^{x_1+1}. (Obecný případ ponecháváme jako cvičení.)

{ Nejprve naplníme proměnné x_q, x_L, x_R počáteční konfigurací stroje M }
$x_q := 0$;
$x_L := 0$;
$x_R := (m^{x_1+1} - 1) \text{ div } (m - 1)$;

{ Nyní simulujeme kroky M, pokud nenastal koncový stav. }
while $x_q < k$ **do begin**

P_1; P_2; ... $P_{k \cdot m}$
end;

{ Uložíme do proměnné x_0 výslednou hodnotu z pásky stroje M. }
$x_0 := \log_m(x_R * (m-1)+1) - 1$

Jeden průchod cyklu **while** může simulovat jeden nebo více kroků stroje M. Laskavý čtenář snadno ověří, že tato simulace je korektní ve smyslu rovnice 2.1. Všimněme si, že jsme k ní potřebovali jediný cyklus **while** . □

Dokázali jsme, že výpočetní síla **while** programu (resp. počítacího stroje schopného vykonat **while** program) je nejméně tak velká, jako výpočetní síla Turingova stroje. Důkaz opačného tvrzení (tj. že Turingův stroj dokáže simulovat každý **while** program) vyplyne z výsledků následující kapitoly.

Cvičení 2.2.11 *Zobecněte* **while** *program z důkazu věty 2.2.10 pro libovolné* $n \geq 1$.

2.3 Primitivní rekurzívní funkce

Primitivní rekurzívní funkce, nazývané též μ-rekurzívní, představují jeden ze způsobů, jak popsat třídu všech částečně vyčíslitelných funkcí nezávisle na výpočetním mechanismu. Byly zavedeny v téže době, kdy vznikal i Turingův stroj. Jak už název napovídá, klíčovou operací při konstrukci těchto funkcí je rekurze.

V této kapitole mimo jiné ukážeme, že výpočetní síla funkcionálních a procedurálních programovacích jazyků je stejná, a že použití rekurze v programech lze vždy nahradit cykly a obráceně.

Definice 2.3.1 *Třída primitivních rekurzívních funkcí je nejmenší třída* $\mathbb{N}^m \to \mathbb{N}$ *funkcí s následujícími vlastnostmi:*

1. *Obsahuje tyto* bázové funkce:
   0 *(nulová funkce)*
   $S(x) = x + 1$ *(následník)*
   $U_i^n(x_1, \ldots, x_n) = x_i$ *(funkce projekce pro* $1 \leq i \leq n$.*)*
2. *Je uzavřená vzhledem na následující operace:*
   - skládání*: jestliže* $h : \mathbb{N}^m \longrightarrow \mathbb{N}$, $g_1 : \mathbb{N}^n \longrightarrow \mathbb{N}$, ..., $g_m : \mathbb{N}^n \longrightarrow \mathbb{N}$ *jsou primitivní rekurzívní funkce, pak i následující funkce* $f : \mathbb{N}^n \to \mathbb{N}$ *je primitivní rekurzívní:*

$$f(x_1, \ldots, x_n) = h(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)) \tag{2.2}$$

   - primitivní rekurze*: jestliže* $h : \mathbb{N}^n \longrightarrow \mathbb{N}$, $g : \mathbb{N}^{n+2} \longrightarrow \mathbb{N}$ *jsou primitivní rekurzívní funkce, pak i následující funkce* $f : \mathbb{N}^{n+1} \longrightarrow \mathbb{N}$ *je primitivní rekurzívní:*

$$\begin{aligned} f(0, x_1, \ldots, x_n) &= h(x_1, \ldots, x_n), \\ f(z+1, x_1, \ldots, x_n) &= g(z, f(z, x_1, \ldots, x_n), x_1, \ldots, x_n), \quad z \geq 0 \end{aligned} \tag{2.3}$$

Následující příklady ukazují způsob konstrukce primitivních rekurzívních funkcí. Přitom se může nulová funkce chápat jako funkce libovolného množství argumentů, vracející vždy hodnotu 0.

Příklad 2.3.2 *Libovolná konstanta, tedy funkce bez argumentů* $const_n() = n$ *vracející vždy tutéž hodnotu, je primitivní rekurzívní funkce:*

$$const_n = \underbrace{S(S(\ldots S}_{n}(0))\ldots)$$

Příklad 2.3.3 *Sčítání* $a(x, y) = x + y$:

$$\begin{aligned} a(0, y) &= y, \\ a(x + 1, y) &= S(a(x, y)). \end{aligned}$$

Tento zápis je zjednodušením pravidla primitivní rekurze; abychom přesně dodrželi podmínky definice 2.3.1, je třeba položit $n = 1$, $h(x_1) = U_1^1(x_1)$, $g(x_1, x_2, x_3) = S(U_2^3(x_1, x_2, x_3))$. *Tím dostaneme*

$$\begin{aligned} a(0, y) &= U_1^1(y), \\ a(x + 1, y) &= S(U_2^3(x, a(x, y), y)). \end{aligned}$$

Příklad 2.3.4 *Násobení* $m(x, y) = x \cdot y$:

$$\begin{aligned} m(0, y) &= 0, \\ m(x + 1, y) &= a(m(x, y), y). \end{aligned}$$

Příklad 2.3.5 *Nezáporný předchůdce* $P(x) = \max(x - 1, 0)$:

$$\begin{aligned} P(0) &= 0, \\ P(x + 1) &= x. \end{aligned}$$

Příklad 2.3.6 *Nezáporné odčítání* $\mathrm{sub}(x, y) = \max(y - x, 0)$ *označujeme též* $y \dot{-} x$:

$$\begin{aligned} \mathrm{sub}(0, y) &= y, \\ \mathrm{sub}(x + 1, y) &= P(\mathrm{sub}(x, y)). \end{aligned}$$

Cvičení 2.3.7 *Uveďte zjednodušený zápis příkladů 2.3.4, 2.3.5 a 2.3.6 do přesné shody s definicí primitivní rekurze.*

Cvičení 2.3.8 *Ukažte, že následující funkce jsou primitivní rekurzívní: (a) umocňování; (b) faktoriál.*

2.3.1 Výpočetní síla primitivní rekurze

Intuitivně se zdá zřejmé, že bychom libovolnou bázovou funkci i výsledek libovolné z uvedených operací dokázali vyčíslit Turingovým strojem. Všimněme si, že každá primitivní rekurzívní funkce je totální, tedy je definována pro věchny m-tice z $\mathbb{N}^m$. Naproti tomu některé funkce vyčíslitelné Turingovým strojem totální nejsou, Turingův stroj se nemusí pro některé hodnoty vstupu zastavit.

Toto pozorování napovídá, že Turingův stroj je výpočetně silnější než aparát primitivních rekurzívních funkcí. Zdá se, že je třeba primitivní rekurzívní funkce posílit něčím, co umožní popsat nekonečně dlouhý výpočet, aby dosáhly síly Turingova stroje. Potřebnou ingrediencí je operace *minimalizace*.

Definice 2.3.9 *Buď $h : \mathbb{N}^{n+1} \to \mathbb{N}$ primitivní rekurzívní, resp. částečně vyčíslitelná funkce.* Minimalizací *funkce h nazýváme funkci $f : \mathbb{N}^n \to \mathbb{N}$, definovanou jako*

$$f(x_1, \ldots, x_n) = \min\{y \in \mathbb{N} | h(x_1, \ldots, x_n, y) = 0\}, \tag{2.4}$$

pokud $h(x_1, \ldots, x_n, z)$ je definována pro $0 \leq z \leq y$. V opačném případě je hodnota $f(x_1, \ldots, x_n)$ nedefinována. Fakt, že funkce f vznikla minimalizací funkce h, zapisujeme zkráceně formou

$$f(x_1, \ldots, x_n) = \mu y[h(x_1, \ldots, x_n, y) = 0].$$

Příklad 2.3.10 *Funkci celočíselného dělení $f(x, y) = \lceil x/y \rceil$ definovanou pro $y > 0$ je možno zapsat jako*

$$f(x, y) = \mu z[m(y, z) \dot{-} x = 0]$$

(zápis $\lceil x \rceil$ označuje zaokrouhlení čísla x nahoru).

Vztah **while** programů a třídy a primitivních rekurzívních funkcí spolu s minimalizací objasňuje následující věta.

Věta 2.3.11 *Buď P* **while** *program s množinou proměnných* **Var** $= \{x_1, \ldots, x_n\}$. *Pak existují funkce $f_1, \ldots, f_n : \mathbb{N}^n \longrightarrow \mathbb{N}$, vytvořené z bázových funkcí operacemi skládání, primitivní rekurze a minimalizace tak, že pro libovolný stav prostředí Σ platí:*

Funkce $f_i(\sigma(x_1), \ldots, \sigma(x_n))$, $1 \leq i \leq n$, jsou definovány $\iff$ $[P](\sigma)$ je definována, a jsou-li všechny definovány, pak

$$f_i(\sigma(x_1), \ldots, \sigma(x_n)) = [P](\sigma)(x_i). \tag{2.5}$$

Důkaz. Předpokládejme, že program P je složen z konstrukcí (i), (ii) a (v) z def. 2.2.1. Konstrukce (iii) a (iv) se dají nahradit pomocí (i), (ii) a (v), viz poznámka 2.2.2 a výsledek cvičení 2.2.5. Ukážeme, že každou z konstrukcí (i), (ii) a (v) lze počítat jen primitivní rekurzívní funkce doplněné minimalizací.

(i) Buď P program složený z jediné instrukce typu (a) $x_i := 0$, (b) $x_i := x_j + 1$, nebo (c) $x_i := x_j$. Pak funkce f_k, $1 \leq k \leq n$, sestrojíme následně:

(a) $f_i(x_1, \ldots, x_n) = 0$,
$f_k(x_1, \ldots, x_n) = U_k^n(x_1, \ldots, x_n)$, $1 \leq k \neq i \leq n$.

(b) $f_i(x_1, \ldots, x_n) = S(U_j^n(x_1, \ldots, x_n))$,
$f_k(x_1, \ldots, x_n) = U_k^n(x_1, \ldots, x_n)$, $1 \leq k \neq i \leq n$.

(c) $f_i(x_1, \ldots, x_n) = U_j^n(x_1, \ldots, x_n)$,
$f_k(x_1, \ldots, x_n) = U_k^n(x_1, \ldots, x_n)$, $1 \leq k \neq i \leq n$.

(ii) Uvažujme program P a funkce $f_i : \mathbb{N}^n \longrightarrow \mathbb{N}$, $1 \leq i \leq n$, ekvivalentní s P ve smyslu rovnice (2.5). Podobně nechť funkce $g_i : \mathbb{N}^n \longrightarrow \mathbb{N}$, $1 \leq i \leq n$, jsou ekvivalentní programu Q. Pak programu $P; Q$ jsou ekvivalentní funkce

$$h_i(x_1, \ldots, x_n) = g_i(f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n)), \quad 1 \leq i \leq n.$$

(v) Uvažujme program P a funkce $f_i : \mathbb{N}^n \longrightarrow \mathbb{N}$, $1 \leq i \leq n$, ekvivalentní s P ve smyslu rovnice (2.5).

Ukážeme, že existují funkce $g_i : \mathbb{N}^n \longrightarrow \mathbb{N}$, $1 \leq i \leq n$, ekvivalentní programu **while** $x_j < x_k$ **do** P ve smyslu rovnice (2.5).

Sestrojíme nejprve funkce $h_i : \mathbb{N}^{n+1} \longrightarrow \mathbb{N}$, $1 \leq i \leq n$, takové, že hodnota $h_i(x_1, \ldots, x_n, z)$ odpovídá obsahu proměnné x_i po z provedeních programu P. Funkce h_i lze obdržet pomocí primitivní rekurze takto:

$$\begin{aligned} h_i(x_1, \ldots, x_n, 0) &= U_i^n(x_1, \ldots, x_n), \\ h_i(x_1, \ldots, x_n, z+1) &= f_i(h_1(x_1, \ldots, x_n, z), \ldots, h_n(x_1, \ldots, x_n, z)), \ z \geq 0, \end{aligned} \tag{2.6}$$

pro všechna i, $1 \leq i \leq n$. Pak pro hledané funkce g_i ekvivalentní našemu **while** programu platí

$$g_i(x_1, \ldots, x_n) = h_i(x_1, \ldots, x_n, w), \quad 1 \leq i \leq n, \tag{2.7}$$

kde w je počet opakování cyklu **while** $x < y$ **do** P. Hodnota w (a tudíž ani hodnota g_i) není definována, pokud se cyklus nezastaví. Tato hodnota je rovna minimálnímu počtu provedení programu P, po kterém přestane platit podmínka $x_j < x_k$, a tedy platí $x_k - x_j \leq 0$. Použitím minimalizace dostáváme

$$w(x_1, \ldots, x_n) = \mu y[h_k(x_1, \ldots, x_n, y) \dot{-} h_j(x_1, \ldots, x_n, y) = 0], \tag{2.8}$$

kde $\dot{-}$ je nezáporné odčítání z příkladu 2.3.6. Dosazením (pomocí operace skládání) do (2.7) dostáváme výsledný tvar funkcí g_i, $1 \leq i \leq n$.

□

Zápis (2.6) je pro přehlednost zjednodušen oproti definici 2.3.1, lze jej však pomocí skládání a projekce snadno převést na formálně přesný zápis primitivní rekurze. Také sémantiku **while** programů, tedy funkce počítané konstrukcemi (i) až (v) z def. 2.2.1, jsme považovali za intuitivně zřejmou. Pokud bychom chtěli důkaz provést precizně, museli bychom sémantiku přesně definovat. Čtenáře odkazujeme např. na [9, 11].

Z předchozí věty vyplývá, že každý algoritmus je možno naprogramovat jen pomocí složených příkazů (čímž implementujeme operaci skládání) a rekurzívního volání procedur nebo funkcí (pomocí kterého naprogramujeme primitivní rekurzi i minimalizaci). Rekurzívní volání funkcí je přitom podstatou funkcionálního programování, například v Lispu. Získali jsme exaktní důkaz, že funkcionální programovací jazyky jsou výpočetně nejméně tak silné jako procedurální.

Cvičení 2.3.12 *Sestrojte pomocí bázových funkcí, skládání, primitivní rekurze a minimalizace následující funkce:*

(a) $f(n) = \lceil \log_2 n \rceil$, $n \geq 1$;
(b) $f(n) = \lfloor \sqrt{n} \rfloor$.

Následující věta završuje sérii výsledků, započatou větami 2.2.10 a 2.3.11, které dávají do souvislosti třídu částečně vyčíslitelných funkcí, **while** programy a primitivní rekurzívní funkce.

Věta 2.3.13 *Každá funkce $f : \mathbb{N}^n \longrightarrow \mathbb{N}$, $n \geq 1$, vytvořené z bázových funkcí operacemi skládání, primitivní rekurze a minimalizace, je vyčíslitelná Turingovým strojem s páskovou abecedou $\{\sqcup, 0, 1\}$.*

Důkaz. Ukážeme, že každou primitivní rekurzívní funkci $f : \mathbb{N}^n \longrightarrow \mathbb{N}$ i každou funkci vzniklou operací minimalizace je možno vyčíslit Turingovým strojem M, který začíná pracovat se vstupem $1^{x_1+1}01^{x_2+1}0 \ldots 01^{x_n+1}$. Náš stroj nikdy nepřekročí levý okraj vstupního slova na pásce o více než jedno políčko vlevo. Pracuje tedy vlastně s jednosměrně nekonečnou páskou.

(i) Nechť $f(x_1, \ldots, x_n) = 0$, pak stroj M přeskočí první znak 1 na pásce a smaže zbytek obsahu pásky.

(ii) Nechť $f(x) = x + 1$, pak použijeme stroj M z příkladu 2.1.10.

(iii) Nechť $f(x_1, \ldots, x_n) = U_i^n(x_1, \ldots, x_n)$, pak M smaže z pásky vše kromě i-tého bloku jedniček.

(iv) Nechť $h : \mathbb{N}^m \longrightarrow \mathbb{N}$, $g_1 : \mathbb{N}^n \longrightarrow \mathbb{N}$, $\ldots$, $g_m : \mathbb{N}^n \longrightarrow \mathbb{N}$ jsou primitivní rekurzívní funkce vyčíslované stroji $M_h, M_{g1}, \ldots, M_{gm}$. Buď $f : \mathbb{N}^n \to \mathbb{N}$ funkce vzniklá jejich složením podle (2.2). Funkci f vypočte stroj M sestrojený takto:

(a) Stroj M vytvoří na pásce dvě kopie vstupu ve tvaru $1^{x_1+1}0 \ldots 01^{x_n+1} \sqcup 1^{x_1+1}0 \ldots 01^{x_n+1}$ a nastaví hlavu na začátek druhé kopie. Zavolá stroj M_{g1} jako podprogram.
(b) Nyní páska obsahuje slovo $1^{x_1+1}0 \ldots 01^{x_n+1} \sqcup 1^{g_1(x_1,\ldots,x_n)+1}$. Stroj M za konec tohoto slova opět zkopíruje původní vstup a zavolá jako podprogram stroj M_{g2}.
(c) Po provedení kroku (b) pro stroje $M_{g2}, \ldots, M_{gm}$ bude páska obsahovat slovo

$$1^{x_1+1}0 \ldots 01^{x_n+1} \sqcup 1^{g_1(x_1,\ldots,x_n)+1} \sqcup 1^{g_2(x_1,\ldots,x_n)+1} \sqcup \ldots \sqcup 1^{g_m(x_1,\ldots,x_n)+1}.$$

Stroj M smaže původní vstup, přepíše znak $\sqcup$ mezi bloky jedniček znakem 0 a zavolá jako podprogram stroj M_h.

(v) Nechť $h : \mathbb{N}^n \longrightarrow \mathbb{N}$, $g : \mathbb{N}^{n+2} \longrightarrow \mathbb{N}$ jsou primitivní rekurzívní funkce počítané stroji M_g a M_h. Buď $f : \mathbb{N}^{n+1} \longrightarrow \mathbb{N}$ funkce sestrojená primitivní rekurzí podle (2.3). Funkci $f(y, x_1, \ldots, x_n)$ vypočte následující stroj M :

(a) M zapíše na pásku slovo $1^{y+1}01^{x_1+1}0 \ldots 01^{x_n+1} \sqcup 1^{x_1+1}0 \ldots 01^{x_n+1}$ a nastaví hlavu na první 1 za mezerou. Zavolá M_h jako podprogram.
(b) Páska nyní obsahuje slovo

$$1^{y+1}1^{x_1+1}0 \ldots 01^{x_n+1} \sqcup 1^{f(0,x_1,\ldots,x_n)+1}.$$

Pro všechna z od 1 do y stroj přidá k poslednímu bloku jedniček prefix 1^z0, sufix $01^{x_1+1}0 \ldots 01^{x_n+1}$ a spustí stroj M_g jako podprogram.
(c) Nakonec stroj smaže z pásky původní vstup a ponechá jen výsledek.

(vi) Turingův stroj počítající funkci vzniklou minimalizací ponecháváme jako cvičení čtenáři.

□

Důsledek 2.3.14 *Následující třídy funkcí jsou totožné:*

- *třída funkcí částečně vyčíslitelných Turingovým strojem;*
- *třída funkcí počítaných* **while** *programy;*
- *třída primitivních rekurzívních funkcí s operací minimalizace.*

Jelikož jsme v důkazu věty 2.2.10 nekladli žádná omezení na páskovou abecedu Turingova stroje, a v důkaze věty 2.3.13 jsme se omezili na páskovou abecedu $\{\sqcup, 0, 1\}$ a jednosměrně nekonečnou pásku, můžeme nyní vyslovit i následující tvrzení:

Důsledek 2.3.15 *Libovolný Turingův stroj je možno simulovat strojem s jednosměrně nekonečnou páskou a s páskovou abecedou* $\{\sqcup, 0, 1\}$.

Cvičení 2.3.16 *Sestrojte Turingovy stroje z důkazu věty 2.3.13, (i) – (vi).*

2.3.2 Primitivní rekurze a for programy*

Během této a předchozí kapitoly jsme měli možnost si všimnout, že jak třída primitivních rekurzívních funkcí, tak třída funkcí vyčíslitelných **for** programy obsahují jen totální funkce. Zdá se navíc, že k popisu cyklu **for** bychom vystačili s primitivní rekurzí (bez minimalizace), a naopak primitivně rekurzívní funkce lze vypočíst cykly **for** . Tuto intuitivní příbuznost primitivních rekurzívních funkcí a **for** programů potvrzuje následující věta.

Věta 2.3.17 *Třída primitivních rekurzívních funkcí a třída funkcí vyčíslovaných* **for** *programy jsou totožné.*

Důkaz.

(i) Ukážeme, že pro každou primitivní rekurzívní funkci $f : \mathbb{N}^n \longrightarrow \mathbb{N}$ existuje **for** program P s proměnnými $\mathbf{Var} \supseteq \{x_0, x_1, \ldots, x_n\}$ tak, že pro libovolný stav prostředí σ platí:

$$[P](\sigma)(x_0) = f(\sigma(x_1), \ldots, \sigma(x_n)). \tag{2.9}$$

K tomu musíme být schopni vypočíst každou bázovou funkci a každou funkci vytvořenou operacemi skládání a primitivní rekurze z definice 2.3.1 pomocí **for** programů. Důkaz pro bázové funkce a operaci skládání je snadný a přenecháváme jej čtenáři. Uvádíme konstrukci pro primitivní rekurzi.

Buďte $g : \mathbb{N}^{n+1} \longrightarrow \mathbb{N}$, $h : \mathbb{N}^{n-1} \longrightarrow \mathbb{N}$, $n > 0$, primitivní rekurzívní funkce a Q, R **for** programy takové, že

$$[Q](\sigma)(x_0) = g(\sigma(x_1), \sigma(x_{n+1}), \sigma(x_2), \ldots, \sigma(x_n)), \tag{2.10}$$

$$[R](\sigma)(x_0) = h(\sigma(x_2), \ldots, \sigma(x_n)). \tag{2.11}$$

Tedy programy P, resp. Q počítají funkce g, resp. h ve smyslu rovnice (2.1). Odlišné pořadí proměnných netvoří rozpor, neboť proměnné lze přejmenovat. Buď $f : \mathbb{N}^n \longrightarrow \mathbb{N}$ funkce definovaná následovně:

$$f(0, x_2, \ldots, x_n) = h(x_2, \ldots, x_n), \tag{2.12}$$

$$f(x_1 + 1, x_2, \ldots, x_n) = g(x_1, f(x_1, x_2, \ldots, x_n), x_2, \ldots, x_n,), \ x_1 \geq 0. \tag{2.13}$$

Ukážeme, že pak existuje program P počítající f a splňující rovnici (2.9). Uvažujme nové proměnné $y_1, \ldots, y_n$ nepoužité v programech Q a R. Pak P může mít následující tvar:

```
y_1 := x_1; ... y_n := x_n;     /* uschováme obsah proměnných x_1, ..., x_n
                                   pro případ, že by jej programy Q, R změnily */
R;                              /* program R uloží do x_0 hodnotu h(x_2, ..., x_n) */
x_1 := 0 ;                      /* nastavíme počítadlo cyklu na 0 */
for y_1 do begin                /* zde při každém průchodu cyklu
                                   má x_0 hodnotu f(x_1, x_2, ..., x_n) */
    y_1 := x_1 ;                /* uschováme hodnotu počítadla cyklu */
    x_2 := y_2 ; ... x_n := y_n ;  /* obnovíme hodnoty uschovaných proměnných */
    x_{n+1} := x_0 ;            /* naplníme x_{n+1} hodnotou
                                   f(x_1, x_2, ..., x_n), viz (2.10), (2.13) */
    Q ;                         /* program Q uloží do x_0 hodnotu
                                   g(x_1, x_{n+1}, x_2, ..., x_n) */
    x_1 := y_1 + 1              /* zvýšíme o 1 počítadlo cyklu */
end
```

(ii) Je nutno pro každý **for** program P s proměnnými $\mathbf{Var} = \{x_1, \ldots, x_n\}$ sestrojit primitivní rekurzívní funkce $f_i : \mathbb{N}^n \longrightarrow \mathbb{N}$, $1 \leq i \leq n$, takové,že

$$f_i(\sigma(x_1), \ldots, \sigma(x_n)) = [P](\sigma)(x_i), \quad 1 \leq i \leq n. \tag{2.14}$$

Tento důkaz je obdobný jako důkaz věty 2.3.11 s tím rozdílem, že konstruujeme v bodě (v) funkce $g_i : \mathbb{N}^n \longrightarrow \mathbb{N}$, $1 \leq i \leq n$, ekvivalentní programu **for** x_j **do** P, $1 \leq j \leq n$. Jelikož nyní známe předem počet opakování cyklu **for** , je hodnota w z rovnice (2.7) rovna x_j, a nepotřebujeme ji vypočítávat pomocí (2.8). Operace minimalizace tudíž není použita.

□

2.3.3 Ackermannova funkce*

V úvodu kapitoly 2.3 jsme je viděli, že každá primitivní rekurzívní funkce je totální. Otevřen zůstal obrácený problém, tedy zda každá totální vyčíslitelná funkce je primitivní rekurzívní, ergo je možno ji vypočíst **for** programem. Odpověď je negativní; například Ackermannova funkce je totální a vyčíslitelná, ale není primitivní rekurzívní. Využívá se mj. v teorii algoritmů při odhadech složitosti (např. pro algoritmus sjednocení ve faktorové množině). Je definována následovně:

$$\begin{aligned} A(1, j) &= 2^j && \text{pro } j \geq 1; \\ A(i, 1) &= A(i - 1, 2) && \text{pro } i \geq 2; \\ A(i, j) &= A(i - 1, A(i, j - 1)) && \text{pro } i \geq 2, j \geq 2. \end{aligned}$$

	$j=1$	$j=2$	$j=3$	$j=4$
$i=1$	2^1	2^2	2^3	2^4
$i=2$	2^2	2^{2^2}	$2^{2^{2^2}}$	$2^{2^{2^{2^2}}}$
$i=3$	2^{2^2}	$2^{2^{\cdot^{\cdot^{\cdot^2}}}\}16}$	$2^{2^{\cdot^{\cdot^{\cdot^2}}}\}2^{2^{\cdot^{\cdot^{\cdot^2}}}\}16}}$	$2^{2^{\cdot^{\cdot^{\cdot^2}}}\}2^{2^{\cdot^{\cdot^{\cdot^2}}}\}2^{2^{\cdot^{\cdot^{\cdot^2}}}\}16}}}$

Tabulka 2.1: Některé hodnoty Ackermannovy funkce.

Při definici je tedy využito dvojité rekurze. Tento způsob definice rekurzívní funkce je korektní, neboť argumenty s proměnnými funkce A na pravé straně jsou vždy stejné nebo nižší než odpovídající argumenty na straně levé. Výpočet bude tedy vždy sestávat z konečného počtu rekurzívních kroků a je možno sestrojit Turingův stroj, který se vždy zastaví a počítá $A(i,j)$. Ackermannova funkce se využívá při složitostní analýze algoritmů. Některé její hodnoty jsou v tabulce 2.1. Bez důkazu uvádíme následující větu:

Věta 2.3.18 *Pro každou primitivní rekurzívní funkci $f(n)$ existuje číslo n_0 tak, že $f(n) < A(n,n)$ pro všechna $n \geq n_0$.*

2.3.4 Pilný bobr*

V této kapitole ukážeme příklad jednoduše definované funkce, která není ani částečně vyčíslitelná.

Uvažujme jednostopý deterministický TS s páskovou abecedou $\{\sqcup, 1\}$ a s $n+1$ stavy, z toho jeden koncový (tedy se součinovou složitostí $2n$). Stroj začne pracovat s prázdnou páskou, zapíše na ni jistý počet jedniček (ne nutně v souvislém bloku) a po konečném počtu kroků se zastaví. Přitom je konstruován tak, aby tento počet jedniček byl co největší. Pro takový TS se vžilo označení *pilný bobr (busy beaver)*. Označme dále $BB(n)$ maximální číslo takové, že existuje pilný bobr s $n+1$ stavy, který zapíše na pásku právě $BB(n)$ jedniček. Jelikož pro dané n existuje jen konečný počet různých TS výše popsaného typu (cca. $8n^2$ včetně páskově symetrických TS), jde při výpočtu $BB(n)$ o stanovení maxima z konečného počtu hodnot.

V tabulce 2.2 uvádíme několik známých hodnot a spodních limitů pro funkci $BB(n)$. Spodní řádek udává, kolik kroků vykoná příslušný pilný bobr, než se zastaví.

Věta 2.3.19 *Pro každou částečně vyčíslitelnou funkci f existuje číslo $x \in \mathbb{N}$ tak, že $f(x) < BB(x)$.*

Důkaz. Povšimněme si nejprve, že platí $BB(n) < BB(n+1)$ pro každé $n \geq 0$ (nový stav vždy umožní zapsat nejméně jednu jedničku navíc). Buď $f(x)$ funkce vyčíslovaná nějakým strojem M_f s páskovou abecedou $\{\sqcup, 1\}$. Pak rovněž funkce

$$g(x) = \max\{f(2x+2), f(2x+3)\} \tag{2.15}$$

je zjevně částečně vyčíslitelná a tedy je vyčíslována nějakým TS M_g s páskovou abecedou $\{\sqcup, 1\}$. (Bez újmy na obecnosti lze předpokládat, že M_f pracuje s jednosměrně nekonečnou páskou. Pak M_g může např. pro vstup x zapsat na pásku $2x+2$, zavolat stroj M_f, po jeho skončení přejít vpravo za konec jeho výstupu, zapsat $2x+3$ a opět zavolat M_f. Konečně porovná $f(2x+2)$ a $f(2x+3)$ a menší z nich smaže.)

n	1	2	3	4	5	6	8
$BB(n)$	1	4	6	13	≥ 4098	$\geq 95,524,079$	$> 8 \cdot 10^{44}$
Kroky	1	5	11	96	11,798,826	8,690,333,381,690,951	?

Tabulka 2.2: Některé známé hodnoty $BB(n)$.

	A	B	C	D	E
$\sqcup$	1B$\rightarrow$	1C$\leftarrow$	1A$\rightarrow$	1A$\rightarrow$	1H$\leftarrow$
1	1A$\rightarrow$	1B$\leftarrow$	1D$\leftarrow$	1E$\leftarrow$	$\sqcup$C$\leftarrow$

Tabulka 2.3: BB, jenž zapíše 4098 jedniček v 11,798,826 krocích (Marxen, Buntrock 1990).

Nechť M_g má m stavů, nepočítaje v to koncový (této konvence se přidržíme až do konce důkazu). Pak pro libovolné číslo x můžeme sestrojit TS M_x s páskovou abecedou $\{\sqcup, 1\}$, který začne pracovat s prázdnou páskou, zapíše na ni $x+1$ jedniček, přesune se na nejlevější z nich (k tomu je třeba $x+2$ stavů) a začne simulovat stroj M_g. Tedy M_x může být zkonstruován pomocí $x+2+m$ stavů. Jelikož M_x splňuje veškeré náležitosti „pilného bobra“ s $x+2+m$ stavy a zapíše na pásku právě $g(x)$ jedniček, platí

$$g(x) \leq BB(x+2+m) \tag{2.16}$$

(neboť $BB(x)$ je definována jako *maximální* počet jedniček, který může pilný bobr na pásku zapsat). Položíme-li nyní $x = m$, obdržíme $g(m) \leq BB(2m+2)$. Podle (2.15) pak

$$\max\{f(2m+2), f(2m+3)\} \leq BB(2m+2) < BB(2m+3) \tag{2.17}$$

a tedy

$$f(2m+3) < BB(2m+3).$$

□

Důsledek 2.3.20 *Funkce $BB(x)$ není částečně vyčíslitelná, problém jejího výpočtu není řešitelný.*

V tabulkách 2.3 a 2.4 jsou uvedeny příklady pilných bobrů počítajících největší známé odhady funkcí $BB(5)$ a $BB(6)$ (není tedy známo, zda neexistují ještě „pilnější“ bobři s pěti, resp. šesti stavy). Stavy strojů jsou označeny A... H, kde H je koncový stav. Výpočet $BB(5)$ lze realizovat i na našem simulátoru Turingova stroje. Další informace o pilných bobrech lze získat na WWW adresách:
`www.math.ucla.edu/~hbe/beaver.html`
`www.drb.insel.de/~heiner/BB/bb_list`

Cvičení 2.3.21 * *Ukažte, že pro funkci f z důkazu věty 2.3.19 platí $f(k) < BB(k)$ pro každé $k \geq 2m+3$.*

Cvičení 2.3.22 *Sestrojte pilného bobra s dvěma stavy (+ koncovým), který zapíše na pásku 4 jedničky.*

	A	B	C	D	E	F
⊔	1B→	1C←	⊔F→	1A→	1H←	⊔A←
1	1A→	1B←	1D←	⊔E←	1F←	⊔C←

Tabulka 2.4: BB, jenž zapíše 95,524,079 jedniček v 8,690,333,381,690,951 krocích (Marxen, Buntrock 1997).

Kapitola 3

Vyčíslitelnost funkcí, rozhodnutelnost problémů

3.1 Univerzální Turingův stroj

Z Turingovy-Churchovy teze, uvedené v úvodní kapitole, plyne, že ke každému konvenčnímu počítači lze sestrojit Turingův stroj, který jej simuluje. Porovnáme-li jednoduchost Turingova stroje se složitostí běžných osobních počítačů, zdá se, že Turingův stroj simulující osobní počítač musel mít ohromné množství stavů a pravidel. Platí skutečně, že čím složitější procedura, tím komplikovanější Turingův stroj potřebujeme k její realizaci?

V této kapitole ukážeme, že odpověď je negativní. Lze zkonstruovat tzv. *univerzální Turingův stroj* (UTS), který umí simulovat *všechny* Turingovy stroje libovolné složitosti (včetně sebe sama). Simulace jednoho výpočetního stroje druhým není nic neobvyklého, např. pro počítače Apple existuje software simulující PC s procesorem Intel a s Windows, takže je na Apple možno spouštět programy pro PC.

Navíc tento UTS vystačí s velmi nízkým počtem páskových symbolů a stavů. Podle Turingovy-Churchovy teze pak tento konkrétní primitivní stroj může simulovat libovolný existující počítač! Z existence UTS plyne důležitý praktický závěr: uvážíme-li, že všechny v praxi užívané číslicové počítače dokáží simulovat Turingův stroj, pak libovolný číslicový počítač může též simulovat libovolný jiný, je-li vybaven dostatečně velkou pamětí.

UTS musí umět simulovat každý TS, jeho vstup tedy musí obsahovat popis Turingova stroje M, který má být simulován. K tomu je nutno popis M jednoznačně zakódovat řetězcem symbolů ze vstupní abecedy Σ_U univerzálního Turingova stroje. Příklad takového zakódování je v následující kapitole.

Zvolme nějaké jednoznačné (injektivní) zakódování Turingových strojů do Σ_U^* a uvažujme posloupnost $\{w_i\}_{i=1}^{\infty}$ kódů všech Turingových strojů v tomto zakódování. Jestliže tuto posloupnost slov nad Σ_U seřadíme např. lexikograficky, lze jednoznačně přiřadit každému TS číslo (index) z $\mathbb{N}$ odpovídající jeho pořadí v posloupnosti. Pak stroj zakódovaný slovem w_i budeme označovat M_i.

Uvažujme kódovaní, které umožňuje pro daný index i zkonstruovat stroj M_i, a obráceně je-li dán stroj M, lze nalézt index i tak, že $M_i = M$. Takové kódování se nazývá efektivní nebo Gödelovo. V dalším textu budeme pracovat pouze s efektivními kódováními.

Efektivní kódování Turingových strojů lze provést i do jednoprvkové abecedou Σ_U, práce s tímto kódováním je však nepohodlná. My v následujícím příkladě použijeme $\Sigma_U = \{c, 0, 1, B, R, L\}$, čímž se konstrukce UTS výrazně zjednoduší. Na základě důsledku 2.3.15 se omezíme na Turingovy stroje s páskovou abecedou $\{0, 1, \sqcup\}$ a s jednosměrně nekonečnou páskou, které v každém kroku posunou hlavu doleva či doprava.

Přechodové zobrazení kódovaného stroje M zapíšeme do tabulky, v níž každý řá-

dek odpovídá jednomu stavu a každý sloupec jednomu páskovému symbolu. Je-li např. $\delta(p,a) = (q,b,\rightarrow)$ pro $p,q \in Q$, $a,b \in \{0,1,\sqcup\}$, pak v políčku o souřadnicích (p,a) je trojice $(q,b,\rightarrow)$. Tuto trojici zakódujeme následovně:

- stavy v Q očíslujeme (počínaje nulou pro počáteční stav, koncový stav bude poslední) a i-tý stav bude zakódován $i+1$ jedničkami;
- dále bude následovat symbol L nebo R pro posuny $\leftarrow$ nebo $\rightarrow$ (lze snadno ukázat, že posun $\downarrow$ k činnosti Turingova stroje není nezbytný);
- na posledním místě je symbol b, tj. 0, 1 nebo $\sqcup$, který má být při použití pravidla zapsán na pásku.

Prázdný symbol $\sqcup$ nahradíme symbolem B, abychom neporušili definici 2.1.1, podle níž vstup pro UTS nesmí obsahovat $\sqcup$. Pokud je pole tabulky prázdné (= pro danou kombinaci stavu a symbolu není definován přechod), zakódujeme jej jedinou nulou.

Jednotlivá pole tabulky od sebe v kódu oddělíme symbolem c a jednotlivé řádky dvojicí cc. Poslední řádek je vždy prázdný – neobsahuje žádné přechody. Celý kód začíná a končí trojicí ccc.

Příklad 3.1.1 *Stroj z příkladu 2.1.12, počítající funkci $f(x,y) = x + y$, bude popsán tabulkou*

	$\sqcup$	0	1
q_0	$--$	$--$	q_1, B, R
q_1	$--$	q_2, B, R	q_2, B, R
q_2	q_F, B, R	$q_2, 1, R$	$q_2, 1, R$
q_F	$--$	$--$	$--$

Kód tohoto stroje je $ccc0c0c11RBcc0c111RBc11RBcc1111RBc111R1c111R1cc0c0c0ccc$.

Věta 3.1.2 *Existuje univerzální Turingův stroj M_U se vstupní abecedou Σ_U, a efektivní kódování všech Turingových strojů s páskovou abecedou $\{0,1,\sqcup\}$ do Σ_U tak, že pro každý takový stroj M, zakódovaný slovem w platí:*

$$M_U(wx) = M(x) \quad \textit{pro libovolné } x \in \{0,1\}^*.$$

Důkaz. Popíšeme neformálně činnost M_U při simulaci libovolného stroje M, zakódovaného výše popsaným způsobem do abecedy $\Sigma_U = \{c,0,1,R,L,\sqcup\}$. Bez újmy na obecnosti (viz kapitola 2.1.2) můžeme uvažovat, že M_U je dvoustopý se vstupem na 1. stopě a že M pracuje s jednosměrně nekonečnou páskou.

- M_U nejprve označí značkami m na druhé stopě první symbol slova w a začátek a konec slova x. První značka na začátku bloku $cc\ldots cc$ označuje aktuální stav stroje M, druhá na začátku slova x pozici jeho hlavy.
- Jeden krok stroje M simuluje M_U takto:
  - přečte symbol slova x nad druhou značkou m;
  - posune první značku nad příslušný podblok $c\ldots c$ označeného bloku $cc\ldots cc$ ve w;
  - vrátí se na začátek slova w a umístí tam na druhou stopu novou značku m;
  - pro každou jedničku v novém stavu stroje M posune novou značku o jeden blok $cc\ldots cc$ vpravo, tím ji nastaví na blok odpovídající novému stavu M;

– zapamatuje si informaci o požadovaném směru posunu a symbolu, který stroj M má zapsat na pásku;
– smaže starou značku označující předchozí stav stroje M;
– přejede vpravo nad druhou značku m, zapíše tam zapamatovaný symbol a posune značku vlevo nebo vpravo.

- Jestliže se M zastaví, pak M_U nejprve smaže z pásky slovo w a ponechá na ní pouze slovo vzniklé simulací stroje M.

- Pokud se M zastavil v koncovém stavu (reprezentovaném úsekem kódu $cc0c0c0cc$), pak i M_U přejde do koncového stavu, v opačném případě se M_U zastaví v jiném stavu než koncovém.

Z uvedeného popisu M_U plyne, že skutečně $M_U(wx) = M(x)$. □

Postup z výše uvedeného důkazu lze demonstrovat na počítačovém simulátoru TS, jenž je vystaven na WWW stránkách autora, pomocí naprogramovaného stroje M_U, jenž se nachází tamtéž v souboru `UNIVERSA.TXT`.

Z existence univerzálního Turingova stroje lze díky větě 2.2.10 usoudit na existenci univerzální **while** programu, schopného simulovat všechny **while** programy (včetně sebe sama). Přímou konstrukci univerzálního **while** programu lze najít např. v [2].

Z věty 2.3.11 plyne dále existence *univerzální částečně vyčíslitelné funkce.* Příslušné tvrzení formuloval poprvé S. Kleene.

Věta 3.1.3 (Kleene) *Existují primitivní rekurzívní funkce g a h takové, že pro každou částečně vyčíslitelnou funkci $f : \mathbb{N} \longrightarrow \mathbb{N}$ lze efektivně najít číslo $i_f \in \mathbb{N}$ s následující vlastností:*

$$f(x) = g(\mu y[h(x, i_f, y) = 0]).$$

Cvičení 3.1.4 *Uvažujte kódování Turingových strojů do abecedy $\{0, 1\}$, a stroj U, který jako vstup obdrží kód w libovolného stroje M. Stroj U vytvoří na pásce dvě kopie w a zavolá jako proceduru UTS. Ten první kopii w chápe jako popis stroje M a druhou kopii jako vstup pro stroj M, tedy začne simulovat činnost M se vstupem w. Co se stane, když stroj U dostane jako vstup kód sebe sama?*

Cvičení 3.1.5 * *Dokažte větu 3.1.3. Všimněte si, že při výpočtu univerzální funkce se jen jedinkrát použije minimalizace.*

3.1.1 Součinová složitost a minimální UTS*

Vzhledem k univerzální výpočetní síle UTS se nabízí otázka, jaký nejjednodušší UTS lze vůbec sestrojit. Tento problém může mít aplikace např v oblasti molekulárních výpočtů nebo mikromechanismů, neboť vlastně udává možnost co nejjednodušší konstrukce univerzálního počítače.

Prvním problémem je měření složitosti. Počet stavů ani počet páskových symbolů nemůže sám o sobě sloužit za uspokojivé měřítko, neboť je známo, že existuje UTS s dvěma stavy (a 18 symboly) a jiný UTS s dvěma páskovými symboly (a 24 stavy). Proto byla jako měřítko zavedena tzv.

$$\textit{součinová složitost} = |\Gamma| \times |Q - F|.$$

Věta 3.1.6 *(Rogožin): Existuje UTS se součinovou složitostí 24 (6×4). Neexistuje UTS se součinovou složitostí menší než 7.*

Problém minimálního UTS je tedy dosud otevřený.

Cenou za nízkou složitost tří výše zmiňovaných strojů je použití mnohem komplikovanějšího kódování, než jsme probrali my. Důmyslným kódováním vstupu lze totiž konstrukci stroje výrazně zjednodušit. Chápeme-li si UTS jako hardware a jeho vstup jako software, pak tento výsledek potvrzuje známý fakt, že čím důmyslnější počítač s bohatší paletou operací, tím kratší program potřebujeme k popisu dané úlohy. Naopak čím jednodušší počítač, tím složitější program jsme nuceni vytvořit. [1]

3.2 Rekurzívní a rekurzívně spočetné množiny

V předchozích kapitolách jsme se zabývali procedurami či algoritmy vyčíslujícími funkce $f : \mathbb{N}^n \longrightarrow \mathbb{N}$. Obecně můžeme namísto $\mathbb{N}^n$ můžeme uvažovat libovolnou spočetnou množinu U možných *zadání* a namísto $\mathbb{N}$ množinu V přípustných *řešení*. Například překladač počítá funkci, jejímž definičním oborem je množina všech syntakticky správných zdrojových programů a oborem hodnot množina všech přeložených cílových kódů.

Nejjednodušším prakticky užívaným oborem hodnot je dvouprvková množina $\{0, 1\}$ nebo $\{ano, ne\}$. Takové funkce slouží k rozhodování, zda prvek $x \in U$ má jistou vlastnost, například zda daný program v Pascalu je syntakticky správně, nebo zda daná n-tice čísel vsazených ve Sportce vyhrává. Uvažujme totální funkci $f : U \longrightarrow \{0, 1\}$ a označme $S = f^{-1}(1)$. Pak výpočet funkce f lze redukovat na rozhodnutí, zda pro dané $x \in U$ platí $x \in S$.

Budeme nyní zkoumat, pro které podmnožiny $S \subseteq U$ existuje algoritmus schopný rozhodnout pro libovolné $x \in U$, zda $x \in S$. K rozhodování budeme používat Turingův stroj pro jeho jednoduchost, a z téhož důvodu se opět omezíme na množinu možných zadání $U = \mathbb{N}^n$, $n \geq 1$. Samozřejmě U může obsahovat i jakákoli jiná zadání, podmínkou pouze je, aby se dala efektivně očíslovat.

Definice 3.2.1 *(i) Množina $A \subseteq \mathbb{N}^n$ se nazývá* rekurzívně spočetná, *pokud existuje TS M přijímající A.*

(ii) Množina $B \subseteq \mathbb{N}^n$ se nazývá rekurzívní, *pokud existuje TS M přijímající B,* a navíc $M(x_1, \ldots, x_n) = \searrow$ pro všechna $(x_1, \ldots, x_n) \in \mathbb{N}^n$.

O stroji splňujícím podmínky definice 3.2.1 (ii) říkáme, že *rozpoznává* množinu A. Každý stroj rozpoznávající množinu A ji současně i přijímá. Takový stroj se pro libovolný vstup $(x_1, \ldots, x_n) \in A$ zastaví v koncovém stavu, a pro $(x_1, \ldots, x_n) \notin A$ se rovněž zastaví, ale ve stavu, který není koncový.

Pokud M množinu A pouze přijímá, ale nerozpoznává, pak se rovněž pro všechny $(x_1, \ldots, x_n) \in A$ zastaví v koncovém stavu, ale pro některé $(x_1, \ldots, x_n) \notin A$ se nemusí vůbec zastavit.

Každá rekurzívní množina je tedy nutně i rekurzívně spočetná. Obrácené tvrzení ovšem neplatí, jak záhy zjistíme. Prozatím studujme vlastnosti rekurzívních množin. Následující i

[1] Viz maxima Stanislawa Lema, že nekonečně velký program nepotřebuje ke své funkci žádný počítač.

některé další výsledky dokážeme jen pro třídu množin $A \subseteq \mathbb{N}$, ale není problém je zobecnit pro N^n, $n \geq 1$. Jak víme, pro libovolné $n \geq 1$ existuje bijekce $\mathbb{N} \leftrightarrow \mathbb{N}^n$, čili množinu $\mathbb{N}^n$ lze efektivně očíslovat.

Věta 3.2.2 *Množina $A \subseteq \mathbb{N}$ je rekurzívní tehdy a jen tehdy, když A a $\overline{A}$ jsou rekurzívně spočetné.*

Důkaz.

„⇒“: Buď A rekurzívní množina, pak existuje TS M, který ji rozpoznává. Tento stroj lze upravit na stroj M' s koncovým stavem $ACCEPT$ a se stavem $REJECT$ v množině Q, viz poznámka 2.1.6. Pak $\mathrm{DOM}(\delta) = (Q - \{ACCEPT, REJECT\}) \times \Gamma$, a přitom $L(M') = L(M)$. Pak stroj M'' získaný z M' záměnou stavů $ACCEPT$ a $REJECT$, tedy $F'' = \{REJECT\}$, rozpoznává množinu $\overline{A}$. Tudíž jak A, tak $\overline{A}$ jsou rekurzívní a tedy i rekurzívně spočetné množiny.

„⇐“: Nechť A a $\overline{A}$ jsou rekurzívně spočetné množiny, pak existuje TS M_1 přijímající A a M_2 přijímající $\overline{A}$. Sestrojme dvoustopý TS M takto: M nejprve zkopíruje vstup x na obě stopy. Pak simuluje střídavě 1 krok stroje M_1 na 1. stopě a 1 krok stroje M_2 na 2. stopě. Pokud M_1 akceptuje, akceptuje i M. Pokud M_2 akceptuje, M se zastaví a neakceptuje. Vzhledem k tomu, že jeden ze strojů M_1 nebo M_2 jistě akceptuje x (protože $x \in A \cup \overline{A}$), M rozpoznává A.

□

Jak se dá očekávat, existují množiny, které nejsou rekurzívní, ale jsou rekurzívně spočetné, a množiny, které nejsou ani rekurzívně spočetné. Toto tvrzení plyne už z faktu, že existuje nespočetně mnoho podmnožin $\mathbb{N}$, ale jen spočetně mnoho Turingových strojů.

Množiny, které nejsou rekurzívně spočetné, se nedají definovat žádným mechanickým postupem (procedurou), a bývají tudíž nesnadno představitelné. Uvedeme příklad takové množin. Uvažujme efektivní očíslování $M_1, M_2, \ldots$ všech Turingových strojů z kapitoly 3.1.

Věta 3.2.3 (Post) *(i) Množina $K = \{i \in \mathbb{N} \mid M_i$ přijímá $i\}$ je rekurzívně spočetná, ale ne rekurzívní.*

(ii) Množina $\overline{K} = \{i \in \mathbb{N} \mid M_i$ nepřijímá $i\}$ není rekurzívně spočetná.

Důkaz.

(ii) Nechť existuje Turingův stroj $M_{\overline{K}}$ přijímající $\overline{K}$. Tento stroj se musí nacházet v našem očíslování všech TS, a platí tedy $M_{\overline{K}} = M_j$ pro jisté $j \in \mathbb{N}$. Zkoumejme chování M_j se vstupem j.

(a) Nechť $j \in \overline{K}$, potom dle definice množiny $\overline{K}$ M_j nepříjímá j a tedy $M_{\overline{K}}$ nepřijímá j (protože $M_{\overline{K}} = M_j$). Pak ovšem $j \notin \overline{K}$, protože $\overline{K}$ je množina přijímaná strojem $M_{\overline{K}}$.

(b) Nechť naopak $j \notin \overline{K}$, potom dle definice $\overline{K}$ M_j přijímá j, tedy i $M_{\overline{K}}$ přijímá j a $j \in \overline{K}$.

V obou případech obdržíme spor, stroj $M_{\overline{K}}$ tudíž nemůže existovat a množina $\overline{K}$ není rekurzívně spočetná.

(i) Ukážeme nejprve, že K je rekurzívně spočetná. Uvažujme TS M přijímající K, jenž pro libovolný vstup $i \in \mathbb{N}$ zkonstruuje kód stroje M_i (viz kapitola 3.1) a simuluje činnost M_i se vstupem i. Pokud M_i přijme i, přijme jej i M.

Dále z věty 3.2.2 plyne, že pokud $\overline{K}$ není rekurzívně spočetná, pak K nemůže být rekurzívní.

□

Cvičení 3.2.4 *Ukažte, že třída rekurzívních množin je uzavřená na sjednocení, průnik a doplněk.*

Cvičení 3.2.5 *Nalezněte množinu, která není rekurzívně spočetná a jejíž doplněk není rekurzívně spočetný.*

3.3 Riceova věta

Formalizujme nyní pojem *vlastnost*, zmíněný stručně v kapitole 3.2. Označme $\mathcal{U} \subseteq 2^{\mathbb{N}}$ třídu všech rekurzívně spočetných podmnožin $\mathbb{N}$.

Definice 3.3.1 *Každá podtřída $\mathcal{S} \subseteq \mathcal{U}$ se nazývá* vlastnost *rekurzívně spočetných množin. Vlastnost $\mathcal{S}$ se nazývá* netriviální, *pokud $\mathcal{S} \neq \emptyset$ a $\mathcal{S} \neq \mathcal{U}$.*

Můžeme si představit, že vlastnost $\mathcal{S}$ je vymezena jistým kritériem rozdělujícím třídu rekurzívně spočetných množin $\mathcal{U}$ na dvě části: množiny patřící do $\mathcal{S}$ (tj. splňující dané kritérium) a množiny nepatřící do $\mathcal{S}$. Všimněme si, že $\mathcal{S}$ je netriviální tehdy a jen tehdy, když $\mathcal{U} - \mathcal{S}$ je netriviální.

Příklad 3.3.2 *Vlastnost $\mathcal{S}_1 = \{\emptyset\}$ je konečná a netriviální. Vlastnost $\mathcal{S}_2 = \{A \subseteq \mathbb{N} : 0 \in A\}$ je nekonečná a netriviální. Vlastnost $\mathcal{S}_3 = \{A \subseteq \mathbb{N} \mid A$ obsahuje všechna sudá čísla a neobsahuje 0$\}$ je triviální (neboť $\mathcal{S}_3 = \emptyset$).*

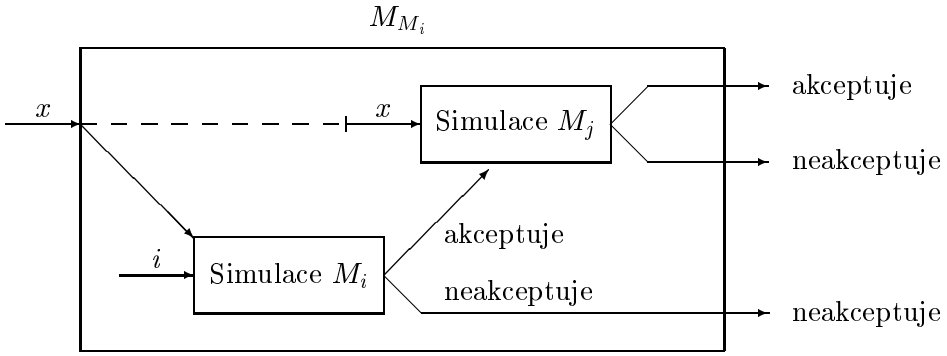
Bude nás nyní zajímat, zda k dané vlastnosti $\mathcal{S}$ lze sestrojit algoritmus, který pro libovolnou rekurzívně spočetnou množinu $A_j \subseteq \mathbb{N}$ rozhodne, zda $A_j \in \mathcal{S}$. Uvažme, že každou takovou množinu A_j lze charakterizovat nějakým Turingovým strojem M_j přijímajícím A_j, a dostaneme přirozeně následující definici.

Definice 3.3.3 *Uvažujme očíslování všech Turingových strojů $M_1, M_2, \ldots$ a označme $A_j \subseteq \mathbb{N}$ množinu přijímanou strojem M_j, $j \geq 0$.*

Vlastnost $\mathcal{S} \subseteq \mathcal{U}$ se nazývá rozhodnutelná, *jestliže existuje Turingův stroj $M_{\mathcal{S}}$ rozpoznávající množinu $\{i \subseteq \mathbb{N} |\ A_i \in \mathcal{S}\}$.*

Důležitá a vcelku překvapivá Riceova věta uvádí na první pohled pesimistické tvrzení o vlastnostech rekurzívně vyčíslitelných množin.

Věta 3.3.4 (Rice) *Každá netriviální vlastnost $\mathcal{S}$ rekurzívně spočetných množin je nerozhodnutelná.*



Obrázek 3.1: Schéma činnosti stroje $M_{M,i}$ z důkazu věty 3.3.4.

Důkaz. Ukážeme sporem, že neexistuje algoritmus rozhodující $\mathcal{S}$. Předpokládejme bez újmy na obecnosti, že $\emptyset \notin \mathcal{S}$ (pokud náhodou $\emptyset \in \mathcal{S}$, vezmeme $\mathcal{U} - \mathcal{S}$, přičemž $\mathcal{S}$ i $\mathcal{U} - \mathcal{S}$ lze rozhodovat tímtéž algoritmem). Jelikož $\mathcal{S}$ je netriviální, existuje jistě alespoň jedna neprázdná množina $A_j \in \mathcal{S}$, $j \geq 1$.

Všimněme si nejprve, že pro libovolné $i \geq 1$ lze sestrojit stroj M_{M_i} (viz obr. 3.1), který pro vstup $x \in \mathbb{N}$ pracuje takto:

1. V první fázi ignoruje svůj vstup x a (na jiné stopě) simuluje činnost stroje M_i se vstupem i.
2. Pokud M_i akceptuje i, pak M_{M_i} začne simulovat M_j se vstupem x.
3. Pokud M_i neakceptuje i, pak M_{M_i} neakceptuje svůj vstup x.

Označme $K = \{i \in \mathbb{N} \mid M_i \text{ akceptuje } i\}$, pak lze psát:

$$M_{M_i} \text{ přijímá } \begin{cases} \text{množinu } A_j \in \mathcal{S}, & \text{pokud } i \in K, \\ \emptyset, & \text{pokud } i \notin K, \end{cases} \tag{3.1}$$

Předpokládejme nyní, že existuje Turingův stroj $M_{\mathcal{S}}$ z definice 3.3.3 rozhodující $\mathcal{S}$. Ukážeme, že pak lze sestrojit stroj M_K rozpoznávající množinu K. Stroj M_K pro vstup i nejprve vypočte index ℓ stroje M_{M_i} v našem efektivním očíslování Turingových strojů, a pak simuluje stroj $M_{\mathcal{S}}$ se vstupem ℓ. Nakonec M_K akceptuje vstup i, pokud $M_{\mathcal{S}}$ akceptuje ℓ.

Jestliže $M_{\mathcal{S}}$ akceptoval ℓ, znamená to, že množina přijímaná strojem M_{M_i} patří do $\mathcal{S}$. Protože podle (3.1) stroj M_{M_i} přijímá buďto $\emptyset$ nebo A_j, a $\emptyset$ nepatří do $\mathcal{S}$, musí M_{M_i} přijímat A_j. Pak opět podle (3.1) $i \in K$. Pokud naopak $M_{\mathcal{S}}$ neakceptoval ℓ, pak M_{M_i} přijímá $\emptyset$, a tedy $i \notin K$.

Stroj M_K tedy skutečně rozpoznává množinu K, což je spor s větou 3.2.3. □

Všechny netriviální vlastnosti rekurzívně spočetných množin jsou tedy nerozhodnutelné. To je dáno složitostí jejich konečné reprezentace, kterou může být Turingův stroj, **while** program a podobně. Pro sebejednodušší vlastnost (např. „být prázdnou množinou“) lze vždy nalézt natolik složitou reprezentaci rekurzívně spočetné množiny, že nelze rozhodnout, zda splňuje kritérium dané vlastnosti.

Na druhé straně si uvědomme, že Riceova věta pouze říká, že k dané vlastnosti $\mathcal{S}$ nelze najít algoritmus rozhodující pro *každou* rek. vyčíslitelnou množinu $A \in \mathcal{U}$, zda $A \in \mathcal{S}$. To nám však nebrání sestrojit podobný algoritmus fungující pro *velmi mnoho* rek.

vyčíslitelných množin. V praktických problémech zpravidla nepotřebujeme vzít v úvahu celou třídu $\mathcal{U}$, ale často jen její nepatrnou část.

3.4 Rozhodnutelné a nerozhodnutelné problémy

Pro další úvahy potřebujeme nejprve upřesnit pojem *problém*. Využijeme označení z kapitoly 3.2. Buď U libovolná spočetná, efektivně očíslovatelná množina možných zadání jisté úlohy, jejichž řešením je odpověď ANO nebo NE. Nazveme ji množinou *případů*. Označme $S \subseteq U$ množinu všech zadání s řešením ANO.

Rozhodovací problém (U, S) je problém zjistit pro libovolné $x \in U$, zda $x \in S$. Příkladem rozhodovacího problému pro $U = \mathbb{N}$ je problém rozhodnout, zda je dané číslo prvočíslem (tedy S je množina všech prvočísel).

Rozhodovací problém se nazývá *rozhodnutelný,* pokud množina S je rekurzívní. V opačném případě se nazývá *nerozhodnutelný*.

3.4.1 Problém zastavení

Následující věta představuje jeden z nejznámějších nerozhodnutelných problémů, od něhož se odvíjí celá řada dalších.

Věta 3.4.1 (Problém zastavení) *Uvažujme efektivní očíslování $M_1, M_2, \ldots$ všech Turingových strojů se vstupní abecedou $\{0, 1\}$. Pro dané $i \in \mathbb{N}$ je nerozhodnutelné, zda $M_i(i) = \searrow$.*

Důkaz. Problém zastavení je charakterizován dvojicí (U, S), kde $U = \mathbb{N}$ a $S = \{i \in \mathbb{N} \mid M_i(i) = \searrow\}$. Předpokládejme, že množina S je rekurzívní a tedy rozpoznávaná Turingovým strojem M_S. Pak je možné sestrojit stroj M_K rozpoznávající množinu $K = \{i \in \mathbb{N} \mid M_i \text{ akceptuje } i\}$ takto:

Stroj M_K pro vstup i nejprve simuluje činnost M_S se vstupem i. Stroj M_S rozhodne, zda $M_i(i) = \searrow$. Pokud M_S akceptuje i, pak M_K simuluje činnost M_i se vstupem i, a akceptuje-li M_i vstup i, pak M_K akceptuje též. Pokud M_S nekaceptuje i, znamená to, že $M_i(i) = \nearrow$, pak M_K se zastaví a neakceptuje i.

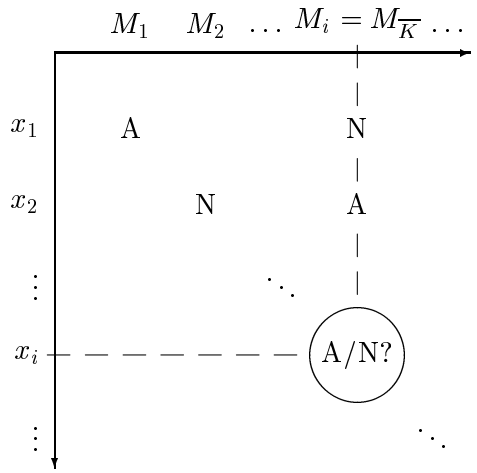
Tím dostáváme spor s větou 3.2.3. □

Důsledek 3.4.2 *Neexistuje algoritmus rozhodující, zda se daný* **while** *program P pro daný vstup x zastaví nebo „zacyklí“.*

Poznámka 3.4.3 Řada otevřených problémů v matematice nebo informatice by byla řešitelná, pokud by problém zastavení byl rozhodnutelný. Uvažme např. jednu z Fermatových vět, která tvrdí, že neexistují přirozená čísla (větší než nula) x, y, z, w tak, že

$$x^{w+2} + y^{w+2} = z^{w+2}. \tag{3.2}$$

Není problém sestrojit Turingův stroj začínající činnost s prázdnou páskou a generující postupně všechny čtveřice $(x, y, z, w) \in \mathbb{N}^4$, který by se zastavil tehdy, platí-li pro (x, y, z, w) rovnice (3.2). Pokud bychom uměli rozhodnout, zda se tento stroj zastaví, byla by tím uvedená Fermatova věta dokázána nebo vyvrácena.



Obrázek 3.2: Schéma důkazu diagonalizací – případy problému příslušnosti.

3.5 Metody dokazování nerozhodnutelnosti

Rozebereme si podrobněji důkazové techniky, jež jsme použili při důkazech v předchozí kapitole. Uvažme, že problém je rozhodnutelný, když existuje rozhodovací algoritmus funkční pro *všechny* jeho případy. Pokud chceme dokázat *rozhodnutelnost* nějakého problému, stačí zkonstruovat příslušný algoritmus.

Pro důkaz *nerozhodnutelnosti* je nutno vyloučit existenci takového algoritmu. Stačí ukázat, že k libovolnému algoritmu lze nalézt vždy alespoň jeden případ, pro který daný algoritmus selže. Nejčastěji k tomu využíváme důkazu sporem: z předpokladu rozhodnutelnosti daného problému dojdeme logicky platným odvozením buďto ke sporu s již dříve dokázaným tvrzením, anebo k paradoxu (tj. k tvrzení, jehož platnost implikuje jeho neplatnost a obráceně).

3.5.1 Metoda diagonalizace

Tato technika redukuje tvrzení o rozhodnutelnosti zkoumaného problému na paradox. V 19. století ji proslavil G. Cantor svým důkazem, že množiny $\mathbb{N}$ a $2^{\mathbb{N}}$ nemají stejnou mohutnost. Příkladem jejícho použití je důkaz věty 3.2.3, kterou je možno přeformulovat následovně:

Věta 3.5.1 *Pro dané $i \in \mathbb{N}$ je nerozhodnutelné, zda M_i akceptuje i.*

Důkaz. Zapišme ve dvourozměrném diagramu na vodorovnou osu všechny stroje M_i, $i \geq 1$, a na svislou osu všechna čísla $j \geq 1$, v průsečíku (i, j) pak bude uvedeno, zda M_i přijímá j. Případy našeho problému jsou tvořeny prvky na diagonále, viz obr. 3.2.

Předpokládejme že by existoval stroj M_K rozhodující náš problém. Nechť tento stroj má stavy *ACCEPT* a *REJECT* z poznámky 2.1.6, pak jejich záměnou obdržíme stroj $M_{\overline{K}}$ z důkazu věty 3.2.3.

Uvažujme libovolné $i \in \mathbb{N}$ na svislé ose, pak stroj $M_{\overline{K}}$ přijímá i právě když stroj M_i nepřijímá i. Tedy chování stroje $M_{\overline{K}}$ se liší od chování *všech* strojů M_i na vodorovné ose, a to právě při zpracování slova i, tedy v situaci zachycené na diagonále. Jelikož však na

vodorovné ose jsou *všechny* Turingovy stroje, stroj $M_{\overline{K}}$ mezi nimi není a tudíž nemůže vůbec existovat! Pak ovšem neexistuje ani stroj M_K. □

Cvičení 3.5.2 * *Dokažte diagonalizací, že neexistuje univerzální primitivní rekurzívní funkce $F_U : \mathbb{N}^2 \longrightarrow \mathbb{N}$ taková, že pro každou primitivní rekurzívní funkci $f : \mathbb{N} \longrightarrow \mathbb{N}$ by existovalo číslo i_f tak, že*

$$F_U(i_f, n) = f(n).$$

3.5.2 Metoda redukce

Uvažujme problém (U_1, S_1), v němž jde o to stanovit pro libovolné $x \in U_1$, zda $x \in S_1$. Pro důkaz jeho nerozhodnutelnosti stačí ukázat, že s využitím algoritmu rozpoznávajícího S_1 bychom dokázali sestrojit algoritmus rozpoznávající nějakou nerekurzívní množinu S_2, čímž obdržíme spor. Takto jsme postupovali v důkazu věty 3.4.1, kde $U_1 = \mathbb{N}$, $S_1 = \{i \in \mathbb{N} \mid M_i(i) = \searrow\}$, a $S_2 = \{i \in \mathbb{N} \mid M_i \text{ akceptuje } i\}$.

Tento postup lze zobecnit: stačí nalézt vhodný problém (U_2, S_2), o němž je dokázáno, že je nerozhodnutelný, a *rekurzívní* zobrazení (tj. vyčíslitelné strojem M_φ, který se zastaví pro každý vstup) $\varphi : U_2 \longrightarrow U_1$ takové, že

$$(x \in S_2) \iff (\varphi(x) \in S_1). \tag{3.3}$$

Buď M_1 stroj rozhodující problém (U_1, S_1). Sestrojme stroj M_2 takto:

1. M_2 se vstupem $x \in U_2$ nejprve simuluje M_φ se vstupem x;
2. poté M_2 simuluje M_1 se vstupem $\varphi(x)$;
3. nakonec M_2 akceptuje x tehdy a jen tehdy, když M_1 akceptuje $\varphi(x)$.

M_2 se vždy zastaví, protože i M_1 a M_φ se vždy zastaví, a vzhledem k (3.3) M_2 rozpoznává S_2, což je spor, neboť množina S_2 není rekurzívní.

Věta 3.5.3 *(Problém tisku) Pro dané $i \in \mathbb{N}$ je nerozhodnutelné, zda $M_i(0) = 0$ (tj. zda výstupem stroje M_i pro vstup 0 bude právě hodnota 0).*

Důkaz. Pro libovolné $i \geq 1$ lze (díky efektivitě našeho očíslování Turingových strojů) sestrojit stroj M_j pracující takto:

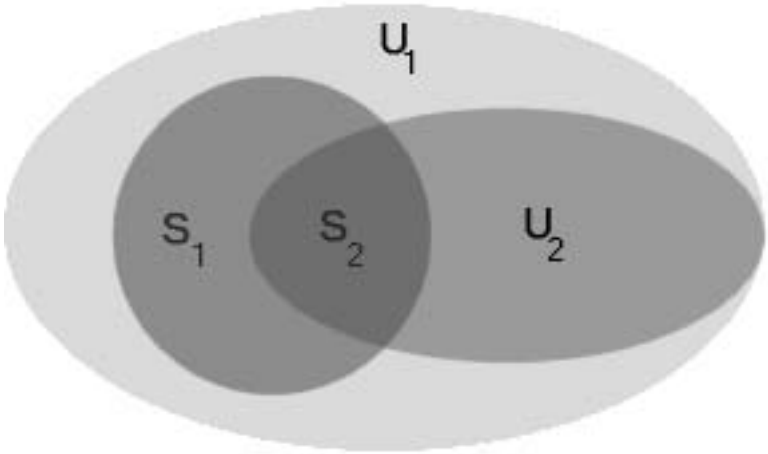
- M_j ignoruje svůj vstup a zapíše na pásku hodnotu i;
- M_j simuluje činnost stroje M_i se vstupem i;
- pokud $M_i(i) = \searrow$, pak M_j zapíše na pásku hodnotu 0 a zastaví se.

Stroj M_j tedy vypíše na pásku 0 tehdy a jen tehdy, když se stroj M_i zastaví se vstupem i. Označme $S_1 = \{i \in \mathbb{N} \mid M_i(0) = 0\}$ a $S_2 = \{i \in \mathbb{N} \mid M_i(i) = \searrow\}$. Buď $\varphi : \mathbb{N} \longrightarrow \mathbb{N}$ zobrazení definované takto:

$$\varphi(i) = j, \text{ kde } j \text{ je index stroje } M_j \text{ popsaného výše.}$$

Všimněme si, že zobrazení φ je rekurzívní a platí

$$(i \in S_2) \iff (\varphi(i) \in S_1).$$



Obrázek 3.3: Redukce problému (U_1, S_1) na užší nerozhodnutelný problém (U_2, S_2).

Předpokládejme, že problém tisku je rozhodnutelný a množina S_1 je rekurzívní. Pak podle výše uvedeného rozboru je rekurzívní i množina S_2, což je spor s větou 3.4.1. □

Z metody redukce vyplývá, že je-li problém (U_2, S_2) nerozhodnutelný, pak i všechny „těžší" problémy obsahující (U_2, S_2) jako svůj podproblém jsou nerozhodnutelné (což je přirozené). Formálně, uvažujme problém (U_1, S_1), kde $U_2 \subseteq U_1$ a platí $S_2 = S_1 \cap U_2$, pak zobrazení φ je identita a problém (U_1, S_1) je nerozhodnutelný. Situaci ilustruje obr. 3.3. Jako příklad poslouží následující problém:

Věta 3.5.4 *(Problém příslušnosti) Pro daný Turingův stroj M_i a $j \in \mathbb{N}$ je nerozhodnutelné, zda M_i akceptuje j.*

Důkaz. Máme $U_1 = \mathbb{N}^2$ a $S_1 = \{(i,j) \in \mathbb{N}^2 \mid M_i \text{ akceptuje } j\}$. Položme $U_2 = \{(i,i) \mid i \in \mathbb{N}\}$ a $S_2 = \{(i,i) \in U_2 \mid M_i \text{ akceptuje } i\}$. Pak $S_2 = S_1 \cap U_2$, přitom (U_2, S_2) je nerozhodnutelný problém z věty 3.5.1. □

Cvičení 3.5.5 *Dokažte, že pro TS M je nerozhodnutelné, zda $M(\epsilon) = \searrow$.*

Cvičení 3.5.6 *Dokažte, že pro TS M je nerozhodnutelné, zda $L(M) = \emptyset$.*

Cvičení 3.5.7 *Ukažte, že pro daný TS M a stav $q \in Q$ je nerozhodnutelné, zda existuje vstup x, při jehož zpracování stroj T může dospět do stavu q.*

Cvičení 3.5.8 *Ukažte, že rozhodnutelnost problému z cvičení 3.5.5 by vedla k vyřešení problému Goldbachova tvrzení (1742), že každé celé číslo větší než 3 je součtem dvou prvočísel (popište konstrukci příslušného TS).*

Obrázek 3.4: Příklad dláždění – obraz „Ještěrky“ C. M. Eschera.

3.6 Některé známé nerozhodnutelné problémy

3.6.1 Problém dláždění

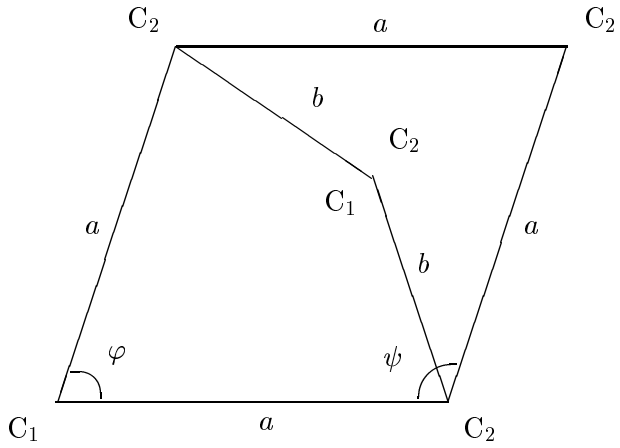
Tento problém se vyskytuje v řadě variant a je zkoumán po několik tisíc let; jde o vyplnění roviny (ev. prostoru) „dlaždicemi“ z nějaké konečné množiny možných typů (speciální pozornost je věnována mnohoúhelníkům a mnohostěnům) bez mezer či překrývání. Variantně, jsou-li dlaždice obarveny, lze požadovat shodu barev na sousedících vrcholech, hranách, stěnách apod. Již Pythagorejské geometrické škole (6. stol. př. n. l.) bylo známo, že existuje pouze jeden pravidelný mnohostěn, jenž zcela vyplní prostor. Existuje však nekonečně mnoho množin s dvěma a více tvary, jež mohou vyplnit rovinu (prostor).

Dláždění se nazývá *periodické*, pokud existuje konečná oblast (složená z konečného počtu dlaždic) tak, že celé dláždění lze získat jejím posouváním bez natáčení (viz obrázek 3.4). V opačném případě se nazývá *aperiodické*. Problém nalezení množin dlaždic tvořících pouze aperiodické dláždění má velmi zajímavá řešení a důsledky (fyzika, chemie, krystalografie).

Cvičení 3.6.1 *Které pravidelné mnohoúhelníky mohou opakováním a natáčením vyplnit celou rovinu?*

Problém dláždění lze tedy formulovat takto: je-li dána konečná množina dlaždic s obarvenými hranami, lze z nich vytvořit dláždění roviny? Přitom není nutno použít všechny dlaždice z této množiny, některé mohou zůstat nevyužity.

V dalším se budeme zabývat speciálním případem, tzv. Wangovými dlaždicemi (dominy), které tvoří čtverce s obarvenými hranami, přičemž rotace ani (zrcadlové) převracení dlaždic není dovoleno. Bylo nalezeno aperiodické dláždění roviny s třinácti dlaždicemi [5]. Je dokázáno, že problém dláždění dominy je nerozhodnutelný. Principy příslušného důkazu si předvedeme na jednodušším tvrzení:



Obrázek 3.5: Penrosovy dlaždice s vrcholy obarvenými barvami C_1 a C_2 (1975), tvořící pouze aperiodické dláždění roviny. $\varphi = 72°$, $\psi = 108°$, $a = (1+\sqrt{5})/2$, $b = 1$.

Věta 3.6.2 *Uvažujme konečné množiny Wangových dlaždic obsahující jednu dlaždici se stejnou barvou na všech čtyřech hranách (např. bílou). Pro danou množinu dlaždic T je nerozhodnutelné, zda existuje takové dláždění roviny, které by obsahovalo konečný počet, avšak nejméně jednu jinou než bílou dlaždici.*

Důkaz. Uvažujme Turingův stroj M, který vždy před akceptováním vstupu smaže obsah pásky (tedy koncová konfigurace je vždy $(q_F, \epsilon, \epsilon)$. Lze snadno ukázat, že každý TS lze modifikovat, aby splnil tyto podmínky, aniž se tím změní přijímaný (rozpoznávaný) jazyk.

Ukážeme, že lze činnost tohoto TS reprezentovat dlážděním, jehož každá řada reprezentuje jednu konfiguraci stroje a sousedící řady reprezentují platný přechod mezi konfiguracemi. Barvy dlaždic jsou označeny symboly nebo skupinami symbolů, bílá barva (neobsahující symbol) je navíc.

Sestrojme množinu dlaždic obsahující následující skupiny:

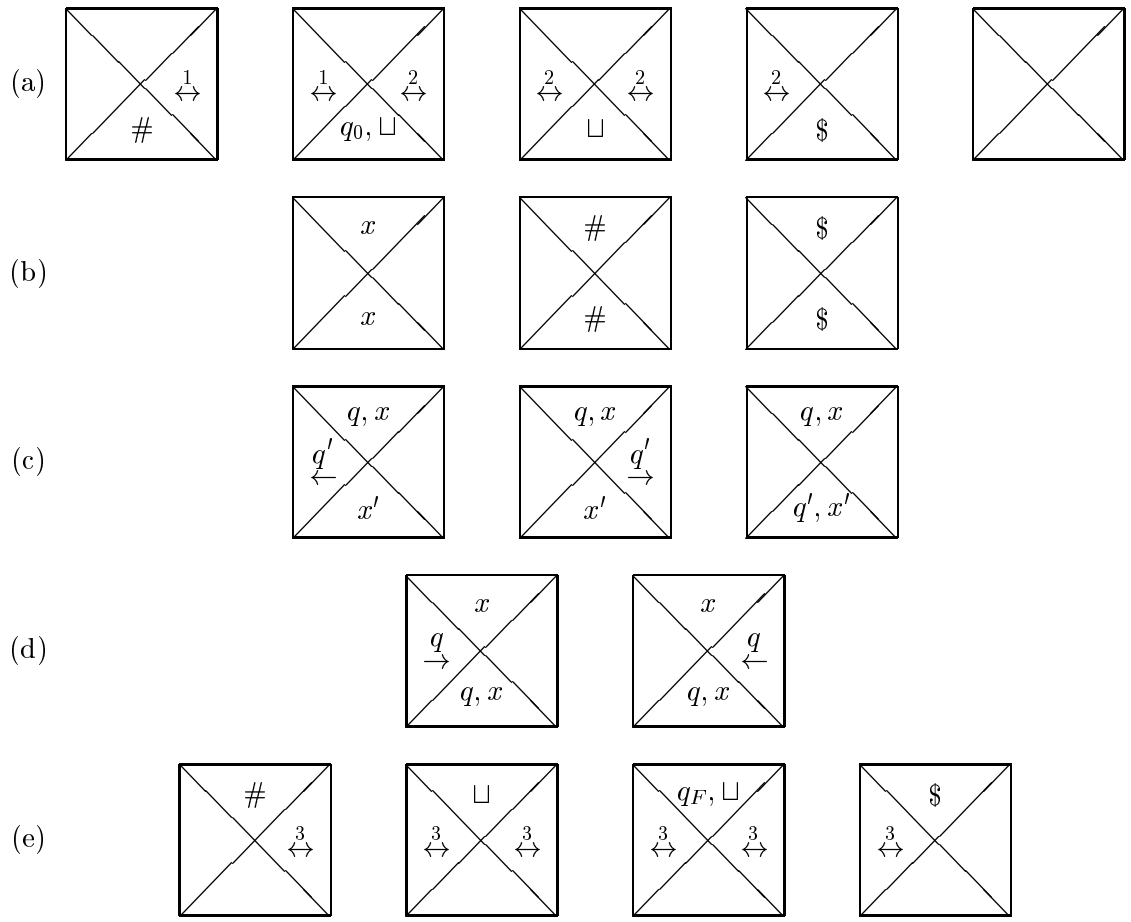
(a) Tyto dlaždice jediné mohou tvořit nejvrchnější řadu „nebílých“ dlaždic v kterémkoli dláždění. Speciální symboly \$, #, $\overset{1}{\leftrightarrow}$ a $\overset{2}{\leftrightarrow}$ nepatří do Γ. Tato řada reprezentuje počáteční konfiguraci, počet dlaždic s mezerou $\sqcup$ musí být dostatečně velký, aby „stačil“ stroji pro jeho příp. přechod do koncové konfigurace.

(b) Tyto dlaždice existují pro každé $x \in \Gamma$. Slouží k přenesení těch částí přepisovaného řetězce, v nichž se nenachází hlava TS, do další konfigurace.

(c) Každému pravidlu stroje $\delta(q,x) = (q', x', \leftarrow)$, $\delta(q,x) = (q', x', \rightarrow)$ nebo $\delta(q,x) = (q', x', \downarrow)$ odpovídá první, druhá nebo třetí dlaždice této skupiny, která slouží k popisu kroku stroje. Všimněme si, že v každé řadě lze použít maximálně jednu z těchto dlaždic.

(d) Pro každý stav $q \in Q$ a pro každé $x \in \Gamma$ existuje dlaždice s příslušnou kombinací $\langle q, x\rangle$. Dokončují jeden krok stroje. Opět v každé řadě lze použít max. jednu z těchto dlaždic.

(e) Tyto dlaždice jediné mohou tvořit koncovou konfiguraci stroje. Speciální symbol $\overset{3}{\leftrightarrow}$ nepatří do Γ.



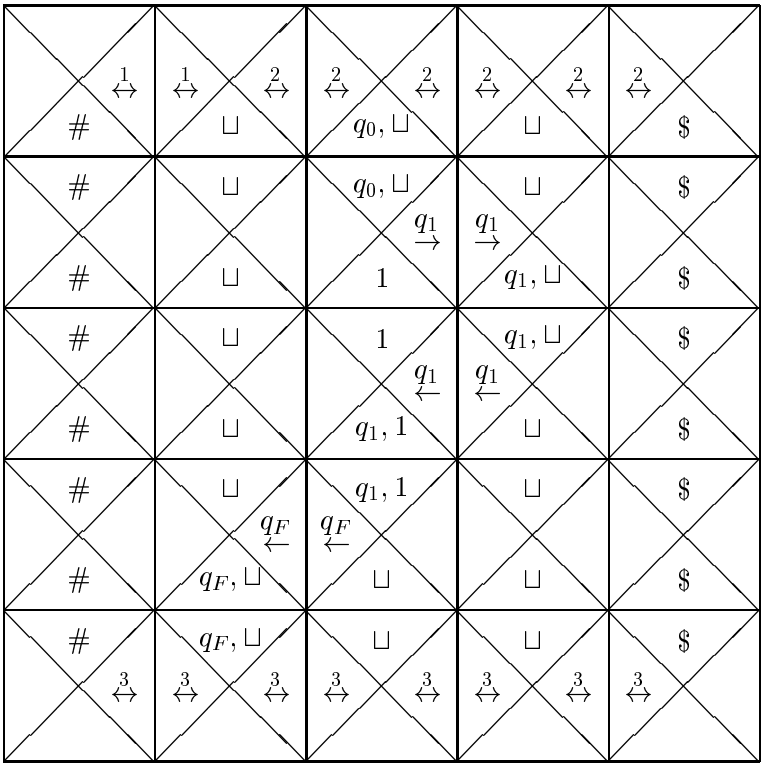
Obrázek 3.6: Skupiny dlaždic (a) až (e) použité v důkaze věty 3.6.2.

Z předvedeného popisu množiny dlaždic T plyne, že výše popsané dláždění roviny (s konečným počtem jiných než bílých dlaždic) existuje tehdy a jen tehdy, když se daný TS M zastaví s prázdnou páskou, což je nerozhodnutelné podle cvičení 3.5.5. □

Příklad 3.6.3 *Uvažujm jednoduchý Turingův stoj řízený pravidly* $\delta(q_0, \sqcup) = (q_1, 1, \rightarrow)$, $\delta(q_1, \sqcup) = (q_1, \sqcup, \leftarrow)$, $\delta(q_1, 1) = (q_F, \sqcup, \leftarrow)$. *Reprezentace jeho činnosti pomocí Wangových dlaždic je na obr. 3.7.*

Cvičení 3.6.4 *Ukažte, že připustíme-li otáčení dlaždic o* 180^o, *bude problém dláždění Wangovými dlaždicemi rozhodnutelný.*

Cvičení 3.6.5 * *Ukažte, že problém dláždění čtvercovými dlaždicemi s obarvenými vrcholy bez možnosti natáčení je nerozhodnutelný. (Návod: ukažte, že problém Wangova dláždění lze redukovat na problém dláždění s obarvenými vrcholy.)*



Obrázek 3.7: Reprezentace činnosti jednoduchého TS pomocí dláždění.

3.6.2 Postův korespondenční problém

Postův korespondenční problém (PKP) má zásadní význam zejména pro studium nerozhodnutelnosti vlastností formálních jazyků. Jeho libovolný případ je definován následovně: Jsou dány dva seznamy n slov nad nějakou konečnou neprázdnou abecedou Σ

$$u = (u_1, u_2, \ldots, u_n), \qquad v = (v_1, v_2, \ldots, v_n).$$

Rozhodněte, zda existuje nějaké $m > 0$ a taková posloupnost čísel $i_1, \ldots, i_m$, kde $1 \leq i_k \leq n$ pro $1 \leq k \leq m$, že platí

$$u_{i_1} \ldots u_{i_m} = v_{i_1} \ldots v_{i_m}.$$

Pokud taková sekvence $(i_1, \ldots, i_m)$ existuje, nazývá se *řešení* daného případu PKP.

Příklad 3.6.6 *1. seznam: (1, 10111, 10), 2. seznam: (111, 10, 0).*
Tento případ má řešení (2,1,1,3): $u_2u_1u_1u_3 = v_2v_1v_1v_3 = 101111110$.

Příklad 3.6.7 *1. seznam: (10, 011, 101), 2. seznam: (101, 11, 011).*
Každé řešení musí začínat 1. slovem. Jelikož v_1 *končí symbolem 1, musí následovat 1. nebo 3. slovo. 1. slovo to být nemůže, neboť máme* $u_1u_1 = 1010$, $v_1v_1 = 101101$. *Musí tedy následovat 3. slovo. Máme tedy* $u_1u_3 = 10101$, $v_1v_3 = 101011$, *tj. dostaneme ekvivalentní situaci, která se stále opakuje. Řešení neexistuje.*

Cvičení 3.6.8 *Rozhodněte, zda následující příklady PKP mají řešení, a pokud ano, nalezněte jej:*

(a) Seznamy (aab, aa), (a, baa).
(b) Seznamy (aab, a), (a, baa).
(c) Seznamy (a, ba, b, ab), (aaa, b, bb, ba).

Zřejmě má-li daný případ alespoň jedno řešení, má jich nekonečně mnoho, vzniklých iterací tohoto řešení. Postův korespondenční problém spočívá v tom, rozhodnout pro libovolný případ, zdali má nebo nemá řešení. Triviální hledání řešení může spočívat v postupném generování všech posloupností $(i_1, \ldots, i_m)$.

Pro důkaz nerozhodnutelnosti PKP definujeme nejprve *modifikovaný PKP,* jehož řešením je sekvence $(1, i_2, \ldots, i_m)$ tak, že platí

$$u_1 u_{i_2} \ldots u_{i_m} = v_1 v_{i_2} \ldots v_{i_m}.$$

Liší se tedy od původního PKP tím, že řešení vždy začíná 1. slovem seznamů.

Lemma 3.6.9 *Kdyby byl PKP rozhodnutelný, pak by byl rozhodnutelný i modifikovaný PKP.*

Důkaz. Pro technickou složitost důkazu odkazujeme na [6]. □

Věta 3.6.10 *Postův korespondenční problém je nerozhodnutelný.*

Důkaz. Ukážeme, že pro libovolný daný TS M (s jediným koncovým stavem q_F) lze sestrojit modifikovaný PKP, který má řešení tehdy a jen tehdy, když $\epsilon \in L(M)$. Konkrétně, nechť $(q_0, \epsilon, \epsilon)$, (q_1, w_1, w_1'), $\ldots$ (q_F, w_k, w_k'), je posloupnost konfigurací daného TS, pak náš mPKP bude mít řešení $\#q_0\#w_1q_1w_1'\#\ldots\#w_kq_Fw_k'\#$, kde $\#$ je spec. symbol, který není v Γ.

Popíšeme nyní konstrukci zmíněného PKP. Slova v seznamech rozdělíme do pěti skupin, slova se stejným pořadím uvedeme vždy pod sebou.

I. První dvojice, jíž musí začínat řešení:

1. seznam	2. seznam
$\#$	$\#q_0\#$

II. skupina:

1. seznam	2. seznam	
$\#$	$\#$	
a	a	pro každé $a \in \Gamma$.

III. skupina:

1. seznam	2. seznam	
qa	bp	pro každé $(q, a, p, b, \rightarrow) \in \delta$,
cqa	pcb	pro každé $(q, a, p, b, \leftarrow) \in \delta$ a $c \in \Gamma$,
$q\#$	$bp\#$	pro každé $(q, \sqcup, p, b, \rightarrow) \in \delta$,
$cq\#$	$pcb\#$	pro každé $(q, \sqcup, p, b, \leftarrow) \in \delta$ a $c \in \Gamma$.

IV. skupina:

1. seznam	2. seznam	
aq_Fb	q_F	pro každé $a, b \in \Gamma$
$aq_F\#$	$q_F\#$	pro každé $a \in \Gamma$
$\#q_Fb$	$\#q_F$	pro každé $b \in \Gamma$

V. skupina: 1. seznam 2. seznam

$q_F\#\#$ $\#$

Ukážeme nyní indukcí, že pokud $(q_0, \epsilon, \epsilon)$, (q_1, w_1, w'_1), $\ldots (q_k, w_k, w'_k)$, kde $q_k \neq q_F$, je posloupnost konfigurací daného TS, pak lze ze seznamů mPKP sestavit (korektním způsobem) slova:

1. seznam $\#q_0\#w_1q_1w'_1\#\ldots\#w_{k-1}q_{k-1}w'_{k-1}\#$
2. seznam $\#q_0\#w_1q_1w'_1\#\ldots\#w_{k-1}q_{k-1}w'_{k-1}\#w_kq_kw'_k\#$

Pro $k = 0$ tvrzení platí vzhledem k I. skupině slov. Předpokládejme, že platí pro nějaké $k > 0$. Pak zbytek 2. slova, který přesahuje 1. slovo, je $w_kq_kw'_k\#$. Abychom mohli rozvíjet slova dále, musíme použít skupinu II. a právě jednu dvojici slov ze skupiny III., která reprezentuje použité pravidlo stroje M. Skupina III. je sestavena tak, že pokud $w_kq_kw'_k \vdash_M w_{k+1}q_{k+1}w'_{k+1}$, tehdy (a jen tehdy) lze obě slova prodloužit do tvaru:

1. seznam $\#q_0\#\ldots\#w_{k-1}q_{k-1}w'_{k-1}\#w_kq_kw'_k\#$
2. seznam $\#q_0\#\ldots\#w_{k-1}q_{k-1}w'_{k-1}\#w_kq_kw'_k\#w_{k+1}q_{k+1}w'_{k+1}\#$

Uvažme nyní, že $q_{k+1} = q_F$. Pak lze pokračovat pouze slovy II. a IV. skupiny, čímž se postupně zkracuje délka 2. slova, až se dostaneme na tvar:

1. seznam $\#q_0\#\ldots\#w_jq_Fw'_j\#$
2. seznam $\#q_0\#\ldots\#w_jq_Fw'_j\#q_F\#$

Zde pak použitím dvojice z V. skupiny nalezneme řešení daného případu mPKP. Vidíme tedy, že popsaný případ mPKP má řešení tehdy a jen tehdy, pokud $\epsilon \in L(M)$. Rozhodnutelnost PKP by podle předchozí věty implikovala rozhodnutelnost mPKP a ta dle předvedené konstrukce rozhodnutelnost problému, zda $\epsilon \in L(M)$. Ten je však nerozhodnutelný a tedy je nerozhodnutelný i PKP. □

Příklad 3.6.11 *Uvažujme TS z obr. 3.7* $M = (\{q_0, q_1, q_2, q_F\}, \emptyset, \{\sqcup, 1\}, \{(q_0, \sqcup, q_1, 1, \rightarrow), (q_1, \sqcup, q_2, 1, \rightarrow), (q_2, \sqcup, q_F, 1, \leftarrow)\}, q_0, \{q_F\})$. *Odpovídající PKP bude obsahovat následující seznamy slov:*

I. skupina:

$u:$	$\#$
$v:$	$\#q_0\#$

II. skupina:

$u:$	$\#$	$\sqcup$	1
$v:$	$\#$	$\sqcup$	1

III. skupina:

$u:$	$q_0\sqcup$	$q_0\#$	$q_1\sqcup$	$q_1\#$	$\sqcup q_2\sqcup$	$1q_2\sqcup$	$\sqcup q_2\#$	$1q_2\#$
$v:$	$1q_1$	$1q_1\#$	$1q_2$	$1q_2\#$	$q_F\sqcup 1$	q_F11	$q_F\sqcup 1\#$	$q_F11\#$

IV. skupina:

$u:$	$\sqcup q_F\sqcup$	$\sqcup q_F1$	$1q_F\sqcup$	$1q_F1$	$\sqcup q_F\#$	$1q_F\#$	$\#q_F\sqcup$	$\#q_F1$
$v:$	q_F	q_F	q_F	q_F	$q_F\#$	$q_F\#$	$\#q_F$	$\#q_F$

V. skupina:

$u:$	$q_F\#\#$
$v:$	$\#$

Řešení bude ve tvaru:

$u:$	$\#$	q_0	$\#$	1	q_1	$\#$	1	1	q_2	$\#$	1	q_F	1	1	$\#$	q_F	1	$\#$	q_F	$\#$	$\#$
$v:$	$\#$	q_0	$\#$	1	q_1	$\#$	1	1	q_2	$\#$	1	q_F	1	1	$\#$	q_F	1	$\#$	q_F	$\#$	$\#$

Cvičení 3.6.12 *Ukažte, že PKP s libovolnými seznamy slov nad jednoprvkovou abecedou je rozhodnutelný.*

3.6.3 Desátý Hilbertův problém*

Byl formulován r. 1900 Davidem Hilbertem jako jeden z 23 klíčových problémů matematiky pro 20. století, jeho dílčí případy byly však známy již dávno předtím. Jeho znění je:

> Pro daný polynom $P(x_1, \ldots, x_n)$ s racionálními koeficienty rozhodněte, zda rovnice $P(x_1, \ldots, x_n) = 0$ má celočíselné nezáporné řešení.

(Taková rovnice se nazývá *diofantická* podle Diophanta z Alexandrie.)

Příklad 3.6.13 *1. Tzv. Pellova rovnice zní $x^2 - 991y^2 - 1 = 0$. V nejmenším známém řešení má x 30 a y 29 míst.*

2. Problém, popsaný Archimédem v dopise Eratosthenovi z Cyrene,[2] *vede na diofantickou rovnici*

$$x^2 - 4729494y^2 - 1 = 0,$$

v jejímž nejmenším známém řešení má x více než 206500 míst.

Důležitost 10. Hilbertova problému spočívá v tom, že je možno na něj převést mnohé jiné problémy z oblasti matematiky a informatiky. Následující věta, kterou uvádíme bez důkazu, udává novou charakterizaci rekurzívně spočetných množin, a to pomocí polynomů.

Věta 3.6.14 (Matijasevič) *Množina $S \subset \mathbb{N}^n$ je rekurzívně spočetná tehdy a jen tehdy, pokud existuje číslo $m \geq 0$ a polynom*[3] *$P(x_1, \ldots, x_n, y_1, \ldots, y_m)$ s racionálními koeficienty tak, že*

$$S = \{(x_1, \ldots, x_n) | \exists (y_1, \ldots, y_m) \in \mathbb{N}^m : P(x_1, \ldots, x_n, y_1, \ldots, y_m) = 0.\} \tag{3.4}$$

K libovolné takové rekurzívně spočetné množině S je navíc možno odpovídající polynom efektivně sestrojit.

Důsledek 3.6.15 *Desátý Hilbertův problém je nerozhodnutelný.*

Důkaz. Uvažujme Turingův stroj M přijímající množinu $S \subset \mathbb{N}^n$, a buď P polynom splňující rovnici (3.4). Pokud by 10. Hilbertův problém byl rozhodnutelný, mohli bychom pro libovolnou n-tici $(x_1, \ldots, x_n) \in \mathbb{N}^n$ rozhodnout, zda rovnice $P(x_1, \ldots, x_n, y_1, \ldots, y_m) = 0$ má řešení, a tedy zda $(x_1, \ldots, x_n) \in S$. To je však spor s tvrzením věty 3.5.4. □

Podle předchozí věty tedy polynomy s racionálními koeficienty mají dostatečnou sílu, aby popsaly svými kořeny kteroukoli rekurzívně spočetnou množinu. Ilustrujeme si tuto jejich překvapivou schopnost na příkladech.

Příklad 3.6.16 *Množinu všech složených čísel lze zapsat jako*

$$C = \{x | \exists y \in \mathbb{N}, z \in \mathbb{N} : x = (y + 2)(z + 2)\},$$

tj. je to množina všech čísel, které lze rozložit na dva činitele velikosti nejméně dva.

[2] Znáte proslulý algoritmus pro hledání prvočísel „Eratosthenovo síto“?

[3] Bylo též dokázáno, že vždy existuje takový polynom stupně max. 4.

Příklad 3.6.17 *Existuje polynom s 26 proměnnými a méně než 100 členy, jehož kořeny jsou všechna prvočísla. Pro jeho složitost ho zde celý neuvádíme a odkazujeme na [5].*

Příklad 3.6.18 *Množině všech nezáporných čísel, které nejsou mocninami dvojky, odpovídá výraz*

$$D = \{x | \exists y \in \mathbb{N}, z \in \mathbb{N} : x = y(2z+3)\},$$

tj. takové číslo je buďto rovno nule (pro $y = 0$), nebo ve svém rozkladu obsahuje lichý člen velikosti nejméně 3 (člen $2z + 3$).

Cvičení 3.6.19 *Vyjádřete pomocí polynomu množinu čísel, jejichž dekadický zápis končí devítkou.*

Cvičení 3.6.20 *Vyjádřete pomocí polynomu množinu čísel, jež (a) jsou násobky tří, (b) nejsou násobky tří. (Nápověda: Obsahuje-li množina S_1 kořeny polynomu P_1 a množina S_2 kořeny P_2, pak množina $S_1 \cup S_2$ obsahuje kořeny $P_1 \cdot P_2$.)*

Cvičení 3.6.21 *Ukažte, že desátý Hilbertův problém je nerozhodnutelný i pro polynomy s celočíselnými koeficienty.*

Univerzální polynom

V předchozích kapitolách jsme charakterizovali rekurzívně spočetné množiny pomocí Turingových strojů. Ukázali jsme, že existují univerzální mechanismy popisující všechny rekurzívně spočetné množiny, jako univerzální Turingův stroj nebo univerzální částečně vyčíslitelná funkce. Lze tedy očekávat, že bude existovat i univerzální polynom s touto vlastností.

Věta 3.6.22 *Existuje polynom $P_u : \mathbb{N}^{n+2} \leftrightarrow \mathbb{N}$ takový, že pro libovolný polynom $P : \mathbb{N} \leftrightarrow \mathbb{N}$ existuje číslo i_P s touto vlastností:*

$$P(x) = 0 \Leftrightarrow \exists (y_1, \ldots, y_n) : (P_u(i_P, x, y_1, \ldots, y_n) = 0).$$

Důkaz. Uvažujme efektivní očíslování $S_1, S_2, \ldots$ všech rekurzívně spočetných množin. Pak množina $S_U = \{(i, x) | x \in S_i\}$ je rovněž rekurzívně spočetná, neboť pro libovolné i lze zkonstruovat S_i a pak (vzhledem k rek. spočetnosti S_i) musí existovat procedura pro určení, zda $x \in S_i$.

K libovolnému polynomu P podle věty 3.6.14 existuje rekurzívně spočetná množina S_P jeho kořenů, jejíž index v uvedeném očíslování označme i_P. Dle předchozího odstavce je množina (i_P, x), kde x jsou kořeny P, rekurzívně spočetná a opět dle věty 3.6.14 existuje tedy polynom P_u argumentů (i_P, x) s výše popsanými vlastnostmi. □

Poznámka 3.6.23 Jak vyplývá ze (zde neuvedeného) důkazu věty 3.6.14, musí existovat polynom P_u stupně 4. Skutečně byl takový nalezen, a to s 58 proměnnými. Snížení počtu proměnných je nutno zaplatit dramatickým nárůstem stupně polynomu (např. pro 13 proměnných je nejnižší známý stupeň cca. $6.6 \cdot 10^{43}$).

Kapitola 4

Rozhodovací problémy formálních jazyků

Rozhodnutelnost problémů formálních jazyků má zásadní praktické důsledky pro aplikace výpočetní techniky. Abychom mohli libovolný problém předložit k řešení počítači, zpravidla jej zapisujeme pomocí formálního (programovacího) jazyka překládaného překladačem. Přitom se vyskytuje řada problémů typu:

- kontroluje daný překladač (tj. formální automat) syntax jazyka v souladu s její definicí (problém ekvivalence)?
- je daný program syntakticky správně (problém příslušnosti)?
- je programovací jazyk L_1 rozšířením jazyka L_2, např. C a C++ (platí $L_1 \subseteq L_2$)?

a řada dalších. Z metody redukce vyplývá, že jsou-li $\mathcal{L}_1 \subseteq \mathcal{L}_2$ dvě třídy formálních jazyků, pak

- veškeré problémy rozhodnutelné v $\mathcal{L}_2$ jsou rozhodnutelné i v $\mathcal{L}_1$;
- veškeré problémy nerozhodnutelné v $\mathcal{L}_1$ jsou nerozhodnutelné i v $\mathcal{L}_2$;
- některé problémy nerozhodnutelné v $\mathcal{L}_2$ mohou být rozhodnutelné v $\mathcal{L}_1$.

Začneme s třídou rekurzívně spočetných jazyků a postupně se budeme zabývat stále užšími třídami. Tak problémy, které se ze začátku ukáží jako rozhodnutelné, jsou nutně rozhodnutelné i v následujících užších třídách. Pokud se nějaký problém ukáže jako nerozhodnutelný v jisté třídě, je nerozhodnutelný i v předchozích širších třídách, ale může se stát rozhodnutelným v následujících užších třídách.

4.1 Turingovy stroje a gramatiky typu 0

Definice 4.1.1 *Jazyk $L \subseteq \Sigma^*$ se nazývá*

(i) rekurzívně spočetný, *pokud existuje TS M se vstupní abecedou Σ přijímající L;*

(ii) rekurzívní, *pokud existuje TS M přijímající L*, a navíc $M(x) = \searrow$ pro všechna $x \in \Sigma^*$.

V souladu s kapitolou 3.2 říkáme, že rekurzívní jazyk je rozpoznáván strojem M. Následující věta objasňuje vztah Turingových strojů a Chomského formálních gramatik.

Věta 4.1.2 *Jazyk je rekurzívně spočetný tehdy a jen tehdy, je-li generován gramatikou typu 0 (dle Chomského).*

Důkaz.

„⇐“: Ukážeme, jak ke každé gramatice $G = (N, T, P, S)$ typu 0 je možno vytvořit TS M tak, že $L(G) = L(M)$. Půjde o dvoustopý nedeterministický TS, na 1. stopě bude uložen vstup x. Na 2. stopě, která je na počátku prázdná, bude stroj M simulovat činnost G při generování slova x. Bude zde vznikající větná forma $w = a_1 a_2 \dots a_k$, kde $a_i \in (N \cup T)$ jsou jednotlivé symboly slova w.

1. M nejprve položí $w := S$, tedy zapíše na 2. stopu symbol S.
2. M nedeterministicky zvolí pozici $i \in \langle 1, |w| \rangle$ a pravidlo $\alpha \to \beta$ z P.
3. Pokud α je prefixem slova $a_i a_{i+1} \dots a_k$, provede M záměnu $\alpha \to \beta$ od pozice i slova w, jinak pokračuje krokem 2.
4. M zkontroluje, zda $x = w$, tedy zda obsahy obou stop se rovnají. V kladném případě se zastaví a akceptuje vstup x, jinak pokračuje krokem 2.

Je zřejmé, že M akceptuje vstup tehdy a jen tehdy, když v gramatice G existuje posloupnost kroků vedoucí k vygenerování slova x, tedy $L(G) = L(M)$.

„⇒“: Ke každému TS $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ sestrojíme gramatiku $G = (N, T, P, S)$ tak, že $L(M) = L(G)$, jejíž pravidla budou simulovat činnost M při akceptování vstupu x. Položme $T := \Sigma$, $N := Q \cup (\Gamma - \Sigma) \cup \{\$, |, \#\}$, kde poslední tři jsou speciální symboly, jež nejsou v Γ. Gramatika bude mít tři skupiny pravidel:

1. Pravidla umožňující odvození $S \Rightarrow^* \$q_0 x|x\#$ pro libovolné $x \in \Sigma^*$.
2. Pravidla simulující kroky stroje M :
   - pro každou pětici $(p, a, q, b, \to) \in \delta$ pravidla $pac \to bqc$ a $pa| \to bq \sqcup |$ pro všechna $c \in \Gamma$;
   - pro každou pětici $(p, a, q, b, \leftarrow) \in \delta$ pravidla $cpa \to qcb$ a $\$pa \to \$q \sqcup b$ pro všechna $c \in \Gamma$;
   - pro každou pětici $(p, a, q, b, \downarrow) \in \delta$ pravidlo $pa \to qb$.
3. Pravidla umožňující odvození $\$x_1 q_F x_2|x\# \Rightarrow^* x$ pro libovolné $x_1, x_2 \in \Gamma^*$, $x \in \Sigma^*$, $q_F \in F$.

Jednotlivé skupiny pravidel jsou užívány v uvedeném pořadí. Gramatika nejprve pravidly skupiny 1 vygeneruje dvě oddělené kopie terminálního slova x. Poté na první kopii pravidly skupiny 2 simuluje činnost M. Pokud dospěje M do koncového stavu, gramatika může pravidly skupiny 3 „vymazat“ z generované větné formy vše kromě 2. kopie slova x. Slovo x tedy může být vygenerováno tehdy a jen tehdy, pokud jej M akceptuje, a tedy $L(M) = L(G)$.

□

4.2 Vlastnosti rekurzívně spočetných jazyků

Pojem *vlastnost* byl pro rekurzívně spočetné množiny zaveden v kapitole 3.3. *Vlastností rekurzívně spočetných jazyků* nazýváme každou podtřídu třídy rek. spočetných jazyků a z Riceovy věty 3.3.4 vyplývá následující tvrzení:

Věta 4.2.1 *Každá netriviální vlastnost rekurzívně spočetných jazyků je nerozhodnutelná.*

Důsledek 4.2.2 *Je nerozhodnutelné, zda daný rekurzívně spočetný jazyk L (a) je prázdný, (b) je konečný, (c) obsahuje dané slovo w (problém příslušnosti), (d) je roven danému rekurzívně spočetnému jazyku L' (problém ekvivalence), ...*

Pro užší třídy jazyků z následujících kapitol je ovšem řada důležitých vlastností rozhodnutelných. Např. problém příslušnosti je rozhodnutelný pro kontextové jazyky, problém prázdnosti je rozhodnutelná pro bezkontextové jazyky atd.

Cvičení 4.2.3 *Dokažte, že pro danou gramatiku G typu 0 je nerozhodnutelné, zda existuje zásobníkový automat přijímající jazyk $L(G)$.*

Cvičení 4.2.4 *Dokažte, že pro danou gramatiku $G = (N, T, P, S)$ typu 0 a daný symbol $A \in N$ je nerozhodnutelné, zda A je nadbytečný. (Pozn. symbol A se nazývá nadbytečný, pokud v G neexistuje žádné odvození $A \Rightarrow^* \omega$ pro $\omega \in T^*$.)*

4.3 Lineárně ohraničené automaty a kontextové gramatiky

Definice 4.3.1 *Lineárně ohraničený automat (LOA) definujeme stejně jako Turingův stroj s následujícími omezeními:*

- *V* Γ *existují speciální znaky* $\$, \# \in (\Gamma - \Sigma)$.
- *Pro* $(p, a, q, b, D) \in \delta$ *musí platit:*

$$\begin{aligned} a = \$ &\implies (b = \$,\ D \neq \leftarrow) \\ a = \# &\implies (b = \#,\ D \neq \rightarrow) \\ a \notin \{\$, \#\} &\implies b \notin \{\$, \#\}. \end{aligned}$$

Omezení v této definici mají za cíl zabránit, aby stroj při své činnosti překročil úsek pásky, na němž je zapsáno vstupní slovo. Na rozdíl od TS tedy jeho paměť není neomezená, ale je ohraničena velikostí vstupního slova. Definice konfigurace a kroku stroje je stejná jako u TS. Mírně se liší definice přijímaného jazyka:

Definice 4.3.2 *Jazyk přijímaný LOA M je množina*

$$L(M) = \{x | (q_0, \epsilon, \$x\#) \vdash_M^* (q_F, w, w')\} \quad \text{pro } q_F \in F,\ x \in \Sigma^*,\ w, w' \in \Gamma^*.$$

Věta 4.3.3 *Jazyk je přijímán lineárně oraničeným automatem tehdy a jen tehdy, je-li generován gramatikou typu 1 (kontextovou).*

Důkaz. Obdobně jako důkaz věty 4.1.2, viz [5, 6]. □

Otevřený problém: jsou nedeterministické LOA silnější než deterministické, tj. přijímají větší třídu jazyků?

Cvičení 4.3.4 *Ukažte, že každý bezkontextový jazyk se dá rozpoznat deterministickým lineárně ohraničeným automatem.*

Cvičení 4.3.5 *Sestrojte nedeterministický lineárně ohraničený automat přijímající jazyk* $\{\alpha\omega\omega | \alpha \in \{0,1\}^*, \omega \in \{0,1\}^+\}$.

Cvičení 4.3.6 *Sestrojte deterministický lineárně ohraničený automat přijímající množinu* $\{2^n, n \in \mathbb{N}\}$. *Čísla budou zakódována jako* 1^{2^n+1}. *Návod: zkuste opakované půlení vstupního slova.*

Následující věta charakterizuje vztah kontextových a rekurzívních jazyků. Označíme-li CS třídu kontextových jazyků a REC třídu rekurzívních jazyků, pak zjevně $CS \subseteq REC$.

Věta 4.3.7 *Existuje rekurzívní jazyk, který není kontextový.*

Důkaz. Uvažujme efektivní očíslování všech kontextových gramatik ve tvaru $G = (N, \{0,1\}, P, S)$. Toto očíslování lze provést pomocí vhodného zakódování podobně, jako jsme očíslovali Turingovy stroje. Uvažujme dále očíslování všech slov z $\{0,1\}^*$, např. podle lexikografického uspořádání.

Uvažujme nyní jazyk $L_O = \{w_i \in \{0,1\}^* | w_i \notin L(G_i)\}$. Podle věty 4.4.3 je tento jazyk rekurzívní, neboť dokážeme sestrojit Turingův stroj, který rozpoznává tento jazyk. Předpokládejme, že L_O je kontextový. Pak musí v našem uspořádání všech kontextových gramatik existovat gramatika G_j tak, že $L(G_j) = L_O$ pro nějaké $j \in \mathbb{N}$. Uvažujme slovo $w_j \in \{0,1\}^*$. Nechť $w_j \in L_O$, potom dle definice L_O $w_j \notin L(G_j)$ a tudíž $w_j \notin L_O$. Nechť naopak $w_j \notin L_O$, potom dle definice L_O $w_j \in L(G_j)$ a tedy $w_j \in L_O$. V obou případech obdržíme spor, v jehož důsledku L_O není v CS. □

4.4 Vlastnosti kontextových jazyků

Rekapitulujme nejprve některé uzávěrové vlastnosti kontextových jazyků.

Lemma 4.4.1 *Třída kontextových jazyků je uzavřená na sjednocení, průnik a doplněk.*

Důkaz. Důkaz uzavřenosti na průnik a sjednocení se provede zcela obdobně jako důkaz prvních dvou tvrzení příkladu 3.2.4. Důkaz uzavřenosti na doplněk je poměrně komplikovaný, byl podán v letech 1987-88 nezávisle R. Szelepcsenyim a N. Immermannem (odkazy viz [5]). □

Připomeňme si, že podle definice třídy kontextových jazyků patří nějaký jazyk L do této třídy, pokud existuje gramatika typu 1, která jej generuje. Tvrzení věty 4.4.1 je tedy třeba rozumět tak, že pro libovolné kontextové gramatiky G_1 a G_2 existují kontextové gramatiky G', G'', G''' tak, že $L(G') = L(G_1) \cup L(G_2)$, $L(G'') = L(G_1) \cap L(G_2)$, $L(G''') = \overline{L(G_1)}$.

Cvičení 4.4.2 *Dokažte, že jsou-li* G_1 *a* G_2 *bezkontextové gramatiky, pak existuje kontextová gramatika* G_{12} *tak, že* $L(G_{12}) = L(G_1) \cap L(G_2)$.

Věta 4.4.3 *Existuje algoritmus (TS, který se vždy zastaví), který pro daný LOA a dané slovo* $w \in \Sigma^*$ *rozhodne, zda* $w \in L(M)$.

Důkaz. LOA se během své činnosti se vstupem x může nacházet v různých konfiguracích, z nichž každá plně popisuje jeho momentální stav. Tyto konfigurace zahrnují:

- aktuální stav $q \in Q$;
- aktuální obsah pásky $w \in \Gamma^{|x|}$;
- pozici hlavy stroje; je možných $|x| + 2$ různých pozic.

Počet všech možných konfigurací je tedy $c_x = |Q| \times |\Gamma|^{|x|} \times (|x| + 2)$. Pokud se deterministický LOA po průchodu $c_x + 1$ konfiguracemi nezastaví, je zřejmé, že musel některou konfigurací projít dvakrát a tedy pracuje v cyklu. Proto stačí simulovat c_x kroků stroje M a pokud se během nich nezastaví, víme, že $x \notin L(M)$. Pokud se naopak M zastaví, platí $x \in L(M)$ tehdy a jen tehdy, když jeho stav je koncový.

Mírně komplikovanější je situace u nedeterministického LOA, kde posloupnosti konfigurací, jimiž může stroj projít, tvoří strom. Tento strom je nutno projít do hloubky c_x; pokud ani jedna jeho větev nevede do koncového stavu, je zřejmo, že $x \notin L(M)$. □

Z věty 4.4.3 plyne, že problém příslušnosti je pro kontextové jazyky (tedy ty, které jsou generovány kontextovými gramatikami) rozhodnutelný. Problém (či vlastnost) prázdnosti však zůstává nerozhodnutelný.

Věta 4.4.4 *Pro kontextovou gramatiku G je nerozhodnutelné, zda $L(G) = \emptyset$.*

Důkaz. Je uveden za větou 4.5.5, jejíž výsledek je ke konstrukci potřeba. □

4.5 Vlastnosti bezkontextových jazyků

Připomeňme si známou pumpovací větu pro bezkontextové jazyky:

Věta 4.5.1 *Pro libovolný bezkontextový jazyk L nad abecedou V existují konstanty p, q s následující vlastností: každé slovo $z \in L$ délky větší než p se dá rozložit na tvar $z = uvwxy$, $u, v, w, x, y \in V^*$, kde $|vwx| \leq q$, $vx \neq \epsilon$, a pro každé $i \geq 0$ platí $uv^iwx^iy \in L$.*

Navíc tyto konstanty p a q lze pro danou bezkontextovou gramatiku nebo konečný automat efektivně nalézt. Z pumpovací věty bezprostředně plynou následující výsledky:

Důsledek 4.5.2 *Pro kontextovou gramatiku $G = (N, T, P, S)$ je rozhodnutelné, zda generuje konečný nebo nekonečný jazyk.*

Důkaz. Podle pumpovací věty je jazyk $L(G)$ nekonečný tehdy a jen tehdy, obsahuje-li alespoň jedno slovo délky větší než p. Ukážeme sporem, že pak musí v $L(G)$ existovat slovo s délkou z intervalu $\langle p + 1, p + q \rangle$.

Nechť $L(G)$ je nekonečný a neobsahuje žádné slovo s délkou z intervalu $\langle p + 1, p + q \rangle$. Uvažujme nejkratší slovo $z \in L(G)$ délky větší než p, které má tudíž délku větší než $p + q$. Pak ovšem $z = uvwxy$, a pro $i = 0$ platí $uv^iwx^iy = uwy \in L(G)$. Délka slova uwy je větší než p, protože vzniklo ze z odebráním vx, a platí $|vwx| \leq q$. Slovo uwy je dále kratší než z, protože $vx \neq \epsilon$. Dostáváme spor s předpokladem, že z je nejkratší slovo z $L(G)$ délky větší než p.

Nyní víme, že $L(G)$ nekonečný tehdy a jen tehdy, obsahuje-li alespoň jedno slovo s délkou z intervalu $\langle p+1, p+q \rangle$. Jelikož problém příslušnosti pro bezkontextové gramatiky je rozhodnutelný, stačí probrat všechna slova z T^* s uvedenou délkou (jichž je konečný počet) a zjistit, zda alespoň jedno patří do $L(G)$. □

Důsledek 4.5.3 *Problém prázdnosti pro bezkontextové jazyky je rozhodnutelný.*

Důkaz. Rozhodnutí o prázdnosti lze učinit následovně: nejprve dle předchozího důkazu prověříme, zda je daný jazyk L nekonečný. Pokud ano, je jistě neprázdný.

Je-li konečný, pak podle pumpovací věty nemůže obsahovat slovo s délkou větší než p. Pak tedy stačí prověřit, zda obsahuje alespoň jedno slovo s délkou nejvýše p. Pokud ne, je prázdný. □

Shrneme některé uzávěrové vlastnosti bezkontextových jazyků (důkazy pro zájemce viz např. [6]).

Lemma 4.5.4 *Třída bezkontextových jazyků (a) je uzavřená na sjednocení a na průnik s regulární množinou, (b) není uzavřená na průnik a na doplněk.*

Pro první důkazy nerozhodnutelnosti některých problémů bezkontextových jazyků využijeme Postův korespondenční problém. Uvažme seznamy $U = (u_1, \ldots, u_n)$ a $V = (v_1, \ldots, v_n)$ libovolného případu PKP nad nějakou abecedou Σ. Definujme bezkontextové gramatiky

$$G_U = (\{S_U\}, \Sigma \cup \{\#, 0, 1\}, P_U, S_U) \quad \text{a} \quad G_V = (\{S_V\}, \Sigma \cup \{\#, 0, 1\}, P_V, S_V),$$

kde $\#$ je speciální symbol, který není v Σ, množina P_U obsahuje pravidla

$$S_U \to u_i S_U i \quad \text{a} \quad S_U \to u_i \# i$$

a množina P_V obsahuje pravidla

$$S_V \to v_i S_V i \quad \text{a} \quad S_V \to v_i \# i$$

pro každé i, $1 \leq i \leq n$, přičemž i je v pravidlech zakódováno jako $1^{i+1}0$. Označme $L_U = L(G_U)$, $L_V = L(G_V)$. Zjevně

$$L_U = \{u_{i_1} u_{i_2} \ldots u_{i_m} \# i_m i_{m-1} \ldots i_1 | m \geq 1\},$$

$$L_V = \{v_{i_1} v_{i_2} \ldots v_{i_m} \# i_m i_{m-1} \ldots i_1 | m \geq 1\}.$$

Věta 4.5.5 *Pro dané bezkontextové gramatiky G_U a G_V je nerozhodnutelné, zda $L(G_U) \cap L(G_V) = \emptyset$.*

Důkaz. Předpokládejme, že tento problém je rozhodnutelný. Pro bezkontextové jazyky L_U a L_V platí, že $L_U \cap L_V = \emptyset$ tehdy a jen tehdy, pokud případ U, V nemá řešení. Pak bychom mohli libovolný případ PKP převést na odpovídající gramatiky G_U a G_V a podle toho, zda $L_U \cap L_V = \emptyset$ či nikoli rozhodnout, zda nemá či má řešení, což je spor s nerozhodnutelností PKP. □

Vyzbrojeni předchozím výsledkem můžeme nyní snadno dokázat větu 4.4.4: z výsledku cvičení 4.4.2 plyne, že existuje kontextová gramatika G_{UV} tak, že $L(G_{UV}) = L(G_U) \cap L(G_V)$. Pokud by bylo rozhodnutelné, zda $L(G_{UV}) = \emptyset$, bylo by rozhodnutelné i zda $L(G_U) \cap L(G_V) = \emptyset$, což je spor s tvrzením věty 4.5.5.

Věta 4.5.6 *Pro daný bezkontextový jazyk L je nerozhodnutelné, zda $\overline{L} = \emptyset$.*

Důkaz. Vzhledem k tomu, že jazyky L_U a L_V jsou deterministické a vzhledem k uzávěrovým vlastnostem deterministických a bezkontextových jazyků je $L_{UV} = \overline{L_U} \cup \overline{L_V}$ bezkontextový jazyk. Pak dle de Morganových pravidel $\overline{L_{UV}} = L_U \cap L_V$ a dle věty 4.5.5 je tedy nerozhodnutelné, zda $\overline{L_{UV}} = \emptyset$. □

Věta 4.5.7 *Pro daný bezkontextový jazyk L a regulární množinu R je nerozhodnutelné, zda (a) $R = L$, (b) $R \subseteq L$.*

Důkaz. Položíme-li $R = \Sigma^*$, pak nerozhodnutelnost těchto vztahů je přímým důsledkem předchozí věty, neboť oba platí tehdy a jen tehdy, když $L = \Sigma^*$. □

Cvičení 4.5.8 *Ukažte, že pro libovolnou bezkontextovou gramatiku G a regulární množinu R je rozhodnutelné, zda $L(G) \subseteq R$.*

Důsledek 4.5.9 *Pro bezkontextové jazyky L_1 a L_2 (každý může být zadán např. gramatikou nebo zásobníkovým automatem) je nerozhodnutelné, zda (a) $L_1 = L_2$, (b) $L_1 \subseteq L_2$.*

Uvážíme-li, že programovací jazyky jsou složitější než bezkontextové, pak z tohoto důsledku mj. vyplývá, že neexistuje univerzální testovací algoritmus, který by pro daný programovací jazyk a daný překladač ověřil, zda překladač neobsahuje chyby.

Věta 4.5.10 *Pro bezkontextové jazyky L_1, L_2 a L_3 je nerozhodnutelné, zda (a) $L_1 \cap L_2$ je bezkontextový jazyk, (b) $\overline{L_3}$ je bezkontextový jazyk.*

Důkaz. Uvažujme jazyky

$$R_{UV} = \{u\#v\$v^R\#u^R | u \in \Sigma^*,\ v \in \{0,1\}^*\},$$

$$S_{UV} = \{y\$z^R | y \in L_U,\ z \in L_V\}.$$

Oba tyto jazyky jsou bezkontextové a deterministické. Nemělo by dělat potíže zjistit, že průnik těchto jazyků $R_{UV} \cap S_{UV}$ je prázdný tehdy a jen tehdy, když případ (U, V) PKP nemá řešení. Jazyk R_{UV} totiž obsahuje jen slova složená ze dvou symetrických polovin, a v průniku tedy mohou být pouze slova tvaru $y\$y^R$, kde $y \in L_U$ a současně $y \in L_V$.

Jazyk $T_{UV} = \overline{S_{UV}} \cup \overline{R_{UV}}$ je bezkontextový. Pro jeho doplněk $\overline{T_{UV}} = S_{UV} \cap R_{UV}$ lze psát

$$\overline{T_{UV}} = \{u\#v\$v^R\#u^R | u = u_{i_1} \dots u_{i_m},\ v = i_m \dots i_1\},$$

kde $(i_1, \dots, i_m)$ je libovolné řešení našeho případu PKP. Předpokládejme, že řešení existuje a označme $\mathcal{J}$ nejkratší takové řešení. Pak i $\mathcal{J}^n$ pro libovolné $n \geq 1$ je řešením. Uvažujme dále regulární množinu $R = \{u^*\#v^*\$(v^R)^*\#(u^R)^*\}$. Platí

$$\overline{T_{UV}} \cap R = \{u^n\#v^n\$(v^R)^n\#(u^R)^n | n \geq 1\},$$

což odpovídá všem řešením PKP ve tvaru $\mathcal{J}^n$). Pro posledně uvedený jazyk $\overline{T_{UV}} \cap R$ lze snadno ukázat, že není bezkontextový.

Víme dále, že třída bezkontextových jazyků je uzavřená na průnik s regulární množinou. Pokud by jazyk $\overline{T_{UV}}$ byl bezkontextový, musel by být bezkontextový i jazyk $\overline{T_{UV}} \cap R$, což je spor. Tedy není bezkontextový ani jazyk $\overline{T_{UV}}$.

Pokud tedy má náš případ PKP řešení, jazyk $\overline{T_{UV}}$ není bezkontextový; pokud řešení neexistuje, jazyk $\overline{T_{UV}}$ je prázdný (a tedy bezkontextový). Tento jazyk je tedy bezkontextový tehdy a jen tehdy, když neexistuje řešení našeho případu PKP, což je však nerozhodnutelné. □

Cvičení 4.5.11 *Gramatika $G = (N, T, P, S)$, kde všechna pravidla z P mají tvar $A \to uBv$ nebo $C \to w$, kde $A, B, C \in N$, $u, v, w \in T^+$, se nazývá* lineární. *Dokažte, že je nerozhodnutelné, zda pro dané lineární gramatiky G_1 a G_2 platí $L(G_1) \cap L(G_2) = \emptyset$.*

Cvičení 4.5.12 *Dokažte, že je nerozhodnutelné, zda daný kontextový jazyk je bezkontextový.*

Cvičení 4.5.13 *Dokažte, že pro danou bezkontextovou gramatiku G a zásobníkový automat M je nerozhodnutelné, zda existuje slovo $\omega \in L(G)$ takové, že $\omega \notin L(M)$.*

4.6 Vlastnosti deterministických jazyků

Deterministické jazyky představují třídu jazyků přijímaných deterministickými zásobníkovými automaty nebo generované $LR(k)$ gramatikami. Víme, že tato třída je vlastní podtřídou třídy bezkontextových jazyků. Nejprve opět shrneme uzávěrové vlastnosti deterministických jazyků.

Lemma 4.6.1 *Třída deterministických jazyků (a) je uzavřená na doplněk a na průnik s regulární množinou, (b) není uzavřená na průnik a na sjednocení.*

Některé problémy nerozhodnutelné pro bezkontextové jazyky jsou rozhodnutelné pro jazyky deterministické.

Věta 4.6.2 *Pro daný deterministický jazyk L a regulární množinu R jsou následující problémy rozhodnutelné:*

(a) Platí $R \subseteq L$?
(b) Je jazyk $\overline{L}$ bezkontextový?
(c) Platí $\overline{L} = \emptyset$?
(d) Platí $L = R$?

Věta 4.6.3 *Pro dané deterministické jazyky L_1 a L_2 jsou následující problémy nerozhodnutelné:*

(a) Platí $L_1 \cap L_2 = \emptyset$?
(b) Je jazyk $L_1 \cap L_2$ bezkontextový?
(c) Je jazyk $L_1 \cup L_2$ deterministický?
(d) Platí $L_1 \subseteq L_2$?

Popis problému	Regulární	LR(k)	Bezkontext.	Kontextová	Typu 0
Je jazyk $L(G)$ prázdný? konečný? nekonečný?	R	R	R	N	N
Je $L(G) = \Sigma^*$?	R	R	N	N	N
Je $L(G_1) = L(G_2)$?	R	R	N	N	N
Je $L(G_1) \subseteq L(G_2)$?	R	N	N	N	N
Je $L(G_1) \cap L(G_2)$ prázdný? konečný? nekonečný?	R	N	N	N	N
Je $L(G) = R$, kde R je konkrétní regulární množina?	R	R	N	N	N
Je jazyk $L(G)$ regulární množina?	T	R	N	N	N
Je $L(G_1) \cap L(G_2)$ jazykem stejného typu?	T	N	N	T	T
Je $L(G_1) \cup L(G_2)$ jazykem stejného typu?	T	N	T	T	T
Je $L(G_1) \cdot L(G_2)$ jazykem stejného typu?	T	N	T	T	T
Je $\overline{L(G)}$ jazykem stejného typu?	T	T	N	T	N

Tabulka 4.1: Tabulka rozhodnutelnosti vybraných problémů pro běžné typy formálních gramatik.

Cvičení 4.6.4 *Dokažte tvrzení věty 4.6.2 (využijte uzávěrových vlastností deterministických a bezkontextových jazyků).*

Cvičení 4.6.5 *Dokažte tvrzení věty 4.6.3 (využijte postupů z kapitoly 4.5).*

Cvičení 4.6.6 *Ukažte, že je nerozhodnutelné, zda daná bezkontextová gramatika generuje deterministický jazyk.*

4.7 Souhrn

Tabulka 4.1 převzatá z [6] a doplněná o novější výsledky shrnuje rozhodnutelnost některých problémů pro nejběžnější typy formálních gramatik. Písmeno R znamená rozhodnutelnost problému, N nerozhodnutelnost, T znamená, že problém je triviální – pro všechny jeho případy je odpověď totožná.

Výsledky cvičení

2.1.7 Zvolíme např. tento postup: při posunu vpravo ve výrazu nalezneme první $\rangle$, bezprostředně vlevo od ní musí být odpovídající $\langle$. Obě přepíšeme pomocným symbolem x a celý cyklus se opakuje. Přitom během přesunu vlevo i vpravo ignorujeme symboly x. Až narazíme během přesunu vpravo na konec výrazu, zkontrolujeme, zda ve výrazu už nezbyly žádné závorky. Pak je výraz v pořádku.

$M = (\{q_0, q_1, q_2, q_F\}, \{\langle, \rangle\}, \{\langle, \rangle, x, \sqcup\}, \delta, q_0, \{q_F\})$, kde

$$\begin{array}{lll} \delta(q_0, \langle)=(q_0, \langle, \rightarrow) & \delta(q_1, x)=(q_1, x, \leftarrow) & \delta(q_2, x)=(q_2, x, \leftarrow) \\ \delta(q_0, x)=(q_0, x, \rightarrow) & \delta(q_1, \langle)=(q_0, x, \rightarrow) & \delta(q_2, \sqcup)=(q_F, \sqcup, \downarrow) \\ \delta(q_0, \rangle)=(q_1, x, \leftarrow) & \delta(q_0, \sqcup)=(q_2, \sqcup, \leftarrow) & \end{array}$$

2.1.12 $M = (\{q_0, q_1, q_2, q_F\}, \{0, 1\}, \{0, 1, \sqcup\}, \delta, q_0, \{q_F\})$, kde

$$\begin{array}{lll} \delta(q_0, 1)=(q_1, \sqcup, \rightarrow) & \delta(q_1, 1)=(q_2, \sqcup, \rightarrow) & \delta(q_2, 1)=(q_2, 1, \rightarrow) \\ \delta(q_1, 0)=(q_2, \sqcup, \rightarrow) & \delta(q_2, 0)=(q_2, 1, \rightarrow) & \delta(q_2, \sqcup)=(q_F, \sqcup, \rightarrow) \end{array}$$

2.1.13 $M = (\{q_0, q_1, q_2, q_3, q_F\}, \{1\}, \{1, \sqcup\}, \delta, q_0, \{q_F\})$.

Nejprve umažeme první jedničku a zkopírujeme zbylých n jedniček na 2. stopu:

$$\delta(q_0, 1, \sqcup)=(q_0, \sqcup, \sqcup, \rightarrow) \qquad \delta(q_1, 1, \sqcup)=(q_1, 1, 1, \rightarrow) \qquad \delta(q_1, \sqcup, \sqcup)=(q_2, \sqcup, \sqcup, \downarrow)$$

Nyní budeme v cyklu přidávat jedničky na 1. stopu a přitom vždy smažeme jednu jedničku na 2. stopě. Počet jedniček na 1. stopě se tím zdvojnásobí:

$$\begin{array}{ll} \delta(q_2, 1, \sqcup)=(q_2, 1, \sqcup, \rightarrow) & \delta(q_3, 1, \sqcup)=(q_3, 1, \sqcup, \leftarrow) \\ \delta(q_2, \sqcup, \sqcup)=(q_3, 1, \sqcup, \leftarrow) & \delta(q_3, 1, 1)=(q_2, 1, \sqcup, \rightarrow) \end{array}$$

Po ukončení cyklu přidáme doleva další jedničku, abychom dostali požadovaný výsledek:

$$\delta(q_3, \sqcup, \sqcup)=(q_F, 1, \sqcup, \downarrow)$$

2.1.14 Označme Q množinu stavů příkladu ze cvičení 2.1.13. Náš stroj označme $M' = (Q', \{1\}, \{1, \sqcup\}, \delta', q_0', \{q_F'\})$, kde $Q' = \{q_0', q_1', q_2', q_3', q_4', q_F'\} \cup \{q_0, q_F\}$.

Náš stroj nejprve přidá na konec pásky jedničku a nastaví se na předposlení pozici původního slova. Tím se dostane do konfigurace $(q_2', 1^n, 11)$. Všimněte si, že řetězec 11 reprezentuje v našem kódování číslo 1:

$$\delta'(q_0', 1, \sqcup)=(q_0', 1, \sqcup, \rightarrow) \qquad \delta'(q_0', \sqcup, \sqcup)=(q_1', 1, \sqcup, \leftarrow)$$

Nyní budeme v cyklu kontrolovat, zda vlevo od hlavy stroje zbývají ještě jedničky. Pokud ano, jednu z nich umažeme, vrátíme se na původní pozici a vyvoláme jako podprogram stroj M z předchozího příkladu. Ten zdvojnásobí číslo reprezentované jedničkami vpravo od hlavy stroje.

$$\begin{array}{lll} \delta'(q'_1,1,\sqcup)=(q'_2,1,\sqcup,\leftarrow) & \delta'(q'_2,\sqcup,\sqcup)=(q'_3,\sqcup,\sqcup,\rightarrow) & \delta'(q'_4,1,1)=(q'_4,1,\sqcup,\rightarrow) \\ \delta'(q'_2,1,\sqcup)=(q'_2,1,1,\leftarrow) & \delta'(q'_3,1,1)=(q'_4,\sqcup,\sqcup,\rightarrow) & \delta'(q'_4,1,\sqcup)=(q_0,1,\sqcup,\downarrow) \end{array}$$

Po ukončení činnosti stroj M opět předá řízení stroji M' : $\delta'(q_F,1,\sqcup) = (q'_2,1,\sqcup,\leftarrow)$. Pokud už vlevo od hlavy stroje nezbývají žádné jedničky, stroj se zastaví: $\delta'(q'_3,1,\sqcup) = (q'_F,1,\sqcup,\downarrow)$.

Z konstrukce strojů M a M' lze snadno vysledovat, že jejich spojením vznikne stroj $M'' = (Q \cup Q', \{1\}, \{1,\sqcup\}, \delta \cup \delta', q'_0, \{q'_F\})$, který počítá požadovanou funkci 2^n.

2.2.5 Uvažujme nové proměnné v, w, z, které nejsou obsaženy v P ani Q, a následující program:

$v := x$; $w := y$; $z := x$;
while $v < w$ **do begin**
    P; $v := w$
end;
while $z \geq w$ **do begin**
    Q; $w := z + 1$
end

2.2.6 Vstup je v proměnné n, výsledek uložíme do proměnné f :

$f := 1$; $x := 0$;
for n **do begin**
  for f **do** /* V cyklu vypočteme $f := f * (x + 1)$; */
    for x **do**
      $f := f + 1$;
  $x := x + 1$ /* Zvýšíme počítadlo cyklu */
end

2.2.7 Uvádíme program pro výpočet $\log_2(n)$, argument je v proměnné n, výsledek v proměnné f :

$f := 0$; $x := 1$;
while $x < n$ **do begin**
    for x **do** /* V cyklu **for** vypočteme $x := x * 2$; */
      $x := x + 1$;
    $f := f + 1$
end

2.2.11 Počáteční konfigurace stroje M je $(q_0, \epsilon, a_1^{x_1} a_2 a_1^{x_2} a_2 \ldots a_2 a_1^{x_n})$, do proměnné x_R je tedy třeba vložit hodnotu m-árního čísla $\underbrace{1\ldots1}_{x_n}2\underbrace{1\ldots1}_{x_{n-1}}2\ldots2\underbrace{1\ldots1}_{x_1}$. Tu vypočteme následujícím podprogramem:

$i := n$; { i je pomocná proměnná }
$x_R := 0$;
for i **do begin**
    $x_R := x_R * m^{x_i+2} + (m^{x_i+1} - 1)$ div $(m-1) + 2 * m^{x_i+1}$;
    $i := i - 1$
end;

2.3.7

$$\begin{aligned} m(0,y) &= 0, \\ m(x+1,y) &= a(U_2^3(x, m(x,y), y), U_3^3(x, m(x,y), y)). \end{aligned}$$

$$\begin{aligned} P(0) &= 0, \\ P(x+1) &= U_1^2(x, P(x)). \end{aligned}$$

$$\begin{aligned} \mathrm{sub}(0,y) &= y, \\ \mathrm{sub}(x+1,y) &= P(U_2^3(x, \mathrm{sub}(x,y), y)). \end{aligned}$$

2.3.8 (a) Umocňování $\mathrm{pw}(x,y) = y^x$

$$\begin{aligned} \mathrm{pw}(0) &= S(0), \\ \mathrm{pw}(x+1,y) &= m(U_2^3(x, \mathrm{pw}(x,y), y), U_3^3(x, \mathrm{pw}(x,y), y)). \end{aligned}$$

(b) Faktoriál $F(n) = n!$

$$\begin{aligned} F(0) &= S(0), \\ F(x+1) &= a(m(x, F(x)), U_2^2(x, F(x))). \end{aligned}$$

2.3.12 Pro funkční hodnotu $f(n) = \lceil \log_2 n \rceil$ lze napsat:

$$f(n) = \min\{y \in \mathbb{N} | y \geq \log_2 n\}, \quad n \geq 1$$

což lze upravit na $\min\{y \in \mathbb{N} | 2^y \geq n\}$. Uvědomme si dále, že platí $a \geq b$ tehdy a jen tehdy, když $\mathrm{sub}(a,b) = 0$, takže dostáváme $f(n) = \mu y[\mathrm{sub}(\mathrm{pw}(y,2), n) = 0]$, kde sub a pw jsou funkce z cvičení 2.3.6 a příkladu 2.3.8.

Takto popsaná funkce f by ovšem pro argument 0 vracela hodnotu 0, což není správně, neboť $f(0)$ není definováno. Musíme tedy do závorky minimalizace přidat ještě člen vyjadřující podmínku $n \geq 1$. Jinými slovy, musíme najít takový výraz $v(n,y)$ aby platilo, že pro $n = 0$ neexistuje žádné y tak, že $v(n,y) = 0$. K tomu by stačilo k nezápornému výrazu $\mathrm{sub}(\mathrm{pw}(y,2)$ v hranatých závorkách přičíst ještě nějaký člen, který bude nula pro $n \geq 1$ a bude větší než nula pro $n = 0$.

Podmínku $n \geq 1$ lze převést na tvar $\mathrm{sub}(n,1) = 0$ a vidíme tedy, že $\mathrm{sub}(n,1)$ je právě požadovaný člen. Výsledný tvar řešení tedy bude

$$f(n) = \mu y[a(\mathrm{sub}(\mathrm{pw}(y,2), n),\ \mathrm{sub}(n,1)) = 0].$$

Aby toto řešení splňovalo přesně podmínky definice 2.3.1, je nutno je ještě formálně upravit pomocí projekcí atd., což je již snadné.

2.3.21 Uvažujme libovolné $k = 2m + 3 + \ell$, kde $\ell \geq 0$. Pozměníme rovnici (2.15) na

$$g(x) = \max\{f(2x + 2), f(2x + 3 + \ell)\}.$$

Stroj M_g, jak je popsán v důkazu věty 2.3.19, bude pak potřebovat nejvýše $m + \ell$ stavů (nové stavy se využijí k připsání ℓ jedniček při výpočtu $f(2x + 3 + \ell)$.) Pak vztah (2.16) přejde na tvar $g(x) \leq BB(x + 2 + m + \ell)$, a z rovnice (2.17) obdržíme

$$\max\{f(2m + 2), f(2m + 3 + \ell)\} \leq BB(2m + 2 + \ell) < BB(2m + 3 + \ell),$$

což znamená, že $f(k) = f(2m + 3 + \ell) < BB(2m + 3 + \ell) = BB(k)$.

2.3.22

	A	B
⊔	1B←	1A→
1	1B→	1H←

3.1.4 Stroj U vytvoří kopii w kódu sebe sama a zavolá jako proceduru UTS. Ten začne simulovat U se vstupem w. Simulovaný stroj U opět vytvoří kopii kódu sebe sama a zavolá UTS, atd. Stroj U se tedy dostane do nekonečného cyklu vytvářením kaskády vzájemně se simulujících modelů sebe sama.

3.2.4 Je nutno ukázat, že pro libovolné dva Turingovy stroje T_1 a T_2, rozpoznávající množiny A_1 a A_2,, existuje

(i) stroj T_3 akceptující $A_1 \cup A_2$,
(ii) stroj T_4 akceptující $A_1 \cap A_2$,
(iii) stroj T_5 akceptující $\overline{A_1}$.

Označme $T_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_{01}, \{q_{F1}\})$ a $T_2 = (Q_2, \Sigma_2, \Gamma_2, \delta_2, q_{02}, \{q_{F2}\})$. Bez újmy na obecnosti lze předpokládat, že $Q_1 \cap Q_2 = \emptyset$. Pak výše uvedené stroje lze zkonstruovat následovně:

(i) $T_3 = (Q_1 \cup Q_2 \cup \{q_0\}, \Sigma_1 \cup \Sigma_2, \Gamma_1 \cup \Gamma_2, \delta, q_0, \{q_{F1}, q_{F2}\})$, kde $q_0 \notin Q_1 \cup Q_2$ a $\delta = \delta_1 \cup \delta_2 \cup \{(q_0, a, q_{01}, a, \downarrow), (q_0, a, q_{02}, a, \downarrow) | a \in \Sigma_1 \cup \Sigma_2\}$.

(ii) T_4 bude dvoustopý a bude pracovat takto: nejprve zkopíruje vstup z 1. na 2. stopu. Pak přejde do stavu q_{01} a bude simulovat činnost T_1 na 1. stopě, přičemž druhou ponechá nezměněnu. Dostane-li se do stavu q_{F1}, vyhledá začátek vstupu na 2. stopě, přejde do stavu $q = 02$ a simuluje činnost T_2 na 2. stopě. Stav q_{F2} je pak koncovým stavem T_4.

(iii) Bez újmy na obecnosti lze předpokládat, že T_1 je deterministický a že $\delta_1(q_{F1}, a)$ je nedefinováno pro libovolné $a \in \Gamma_1$. Pak $T_5 = (Q_1, \Sigma_1, \Gamma_1, \delta, q_0, \{q_{F1}\})$, kde

$$\begin{array}{lcl} \delta(q,a) & = & (p, b, D), \text{ pokud } \delta_1(q,a) = (p,b,D) \text{ a } p \neq q_{F1}, \\ \delta(q,a) & = & (q_{F1}, a, \downarrow), \text{ pokud } \delta_1(q,a) \text{ je nedefinováno,} \\ \delta(q,a) & & \text{je nedefinováno, pokud } \delta_1(q,a) = (q_{F1}, b, D) \text{ pro libovolné} \\ & & b \in \Gamma_1,\ D \in \{\leftarrow, \downarrow, \rightarrow\}. \end{array}$$

Pak zjevně T_5 dojde do koncového stavu tehdy a jen tehdy, když se T_1 zastaví mimo koncový stav, tedy T_5 rozpoznává $\overline{A_1}$.

Poznámka: Promyslete si, proč jsme pro konstrukci (iii) potřebovali determinismus stroje T_1.

3.2.5 Je možné postupovat podobně jako u důkazu rekurzívní nespočetnosti množiny $\overline{K}$, provedeného diagonalizací. Zde ovšem musíme čísla „neakceptovatelná“ daným TS vložit jak do množiny, tak do jejího doplňku, tedy provést něco jako dvojitou diagonalizaci. Uvažujme například množinu

$$\mathcal{K} = \{2i \mid M_i \text{ akceptuje } 2i\} \cup \{2i-1 \mid M_i \text{ nekaceptuje } 2i-1\},$$

kde $M_1, M_2 \ldots$ je efektivní očíslování Turingových strojů (se vstupní abecedou $\{0,1\}$).

(a) Předpokládejme, že $\mathcal{K}$ je rek. spočetná a tedy přijímaná strojem M_j, $j \geq 0$. Pak $2j-1 \in \mathcal{K}$ tehdy a jen tehdy, když M_j neakceptuje $2j-1$, a tedy $2j-1 \notin \mathcal{K}$, spor.

(b) Naopak pokud $\overline{\mathcal{K}}$ je rek. spočetná, je přijímána strojem M_k pro nějaké $k > 0$. Pak $2k \in \overline{\mathcal{K}}$ právě když $2k \notin \mathcal{K}$, což platí tehdy a jen tehdy, když M_k neakceptuje $2k$, a tedy $2k \notin \overline{\mathcal{K}}$, opět spor.

3.5.2 Je-li funkce F_U primitivní rekurzívní, pak je jistě taková i funkce

$$g(n) = F_U(n,n) + 1 = S(F_U(U_1^1(n), U_1^1(n)))$$

(vytvořená z F_U použitím skládání, projekce a následníka). Jelikož funkce F_U má být univerzální, musí existovat číslo $i_g \in \mathbb{N}$ tak, že

$$F_U(i_g, n) = g(n) \text{ pro všechna } n \in \mathbb{N}.$$

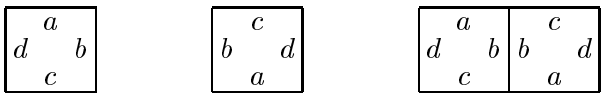
Pro $n = i_g$ ovšem dostáváme $F_U(i_g, i_g) = g(i_g) = F_U(i_g, i_g) + 1$, což je spor.

3.5.5 Množina případů tohoto problému je obsažena v množině případů problému zastavení, takže je „snazší“ než problém zastavení. Nelze tudíž jako zobrazení φ použít identitu.

Uvažujme libovolný TS M a libovolné slovo ω. Sestrojme stroj M' takto: M' nejprve napíše na prázdnou pásku slovo ω a pak simuluje stroj M při zpracování tohoto slova. Stroj M' se tedy zastaví se vstupem ϵ tehdy a jen tehdy, když stroj M se zastaví se vstupem ω. Předpokládejme nyní, že je rozhodnutelné, zda $M'(\epsilon) = \searrow$. Pak je dle předchozího rozhodnutelné i zda $M(\omega) = \searrow$, což je spor s větou o nerozhodnutelnosti problému zastavení.

3.5.6 Připomeňme nejprve, že pro libovolný TS M a libovolné slovo ω je nerozhodnutelné, zda M akceptuje ω (problém příslušnosti). Nyní sestrojme stroj M' takto: M' nejprve zkontroluje, zda na pásce je slovo ω (pokud ne, tak se zastaví a neakceptuje) a pak simuluje stroj M při zpracování tohoto slova. Stroj M' tedy může akceptovat jedině vstup ω, a to ještě jen tehdy, pokud $\omega \in L(M)$. Pak $L(M') \neq \emptyset$ tehdy a jen tehdy, když $\omega \in L(M)$. Předpokládejme nyní, že je rozhodnutelné, zda $L(M') = \emptyset$. Pak je dle předchozího rozhodnutelné i zda $\omega \in L(M)$, což je spor se nerozhodnutelností problému příslušnosti.

Jiné možné řešení příkladu je převést TS na gramatiku typu 0 a užít větu o nerozhodnutelnosti problému prázdnosti pro gramatiky typu 0 (protože tento problém je nerozhodnutelný pro kontextové gramatiky, jak jsme ukázali ve větě 4.4.4, a třída kontextových gramatik je podtřídou třídy gramatik typu 0).



(a) (b) (c)

Obrázek 4.1: Překlápění Wangových dlaždic o 180^o.

3.5.7 Uvažujme, že tento problém je rozhodnutelný, tj. máme algoritmus k jeho řešení. Použijeme-li tento algoritmus postupně na všechny koncové stavy stroje M, můžeme rozhodnout, zda M může pro nějaký vstup dospět do některého koncového stavu, a tedy zda platí $L(M) = \emptyset$ nebo ne. To je ovšem dle výsledku předchozího příkladu nerozhodnutelné.

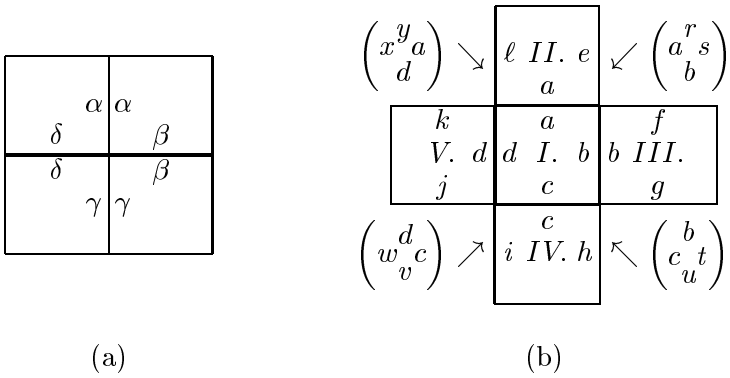
3.5.8 Snadno sestrojíme TS T_1, který pro daný vstup n rozhodne, zda n je prvočíslo. Dále uvažme TS T_2, který pro daný vstup m rozhodne, zda m je součtem dvou prvočísel. T_2 např. může postupně generovat všechny dvojice $(m, m-k)$, kde $2 \leq k \leq m-2$ a pomocí stroje T_1 rozhodovat, zda k a $m-k$ jsou prvočísla. Konečně si představme stroj T_3, který začne pracovat s prázdným vstupem, systematicky generuje všechna čísla větší než 3, pro každé z nich strojem T_2 prověří, zda je součtem dvou prvočísel, a pokud ne, zastaví se. Všechny tyto stroje jistě existují. Pak stroj T_3 se s prázdným vstupem zastaví tehdy a jen tehdy, když Goldbachovo tvrzení neplatí.

3.6.1 Jsou pouze tři - rovnostranný trojúhelník, čtverec a pravidelný šestiúhelník. Všechny ostatní pravidelné mnohoúhelníky je nutno doplnit ještě jiným útvarem, aby se zaplnila celá rovina. Výsledek je intuitivně zřejmý, exaktní důkaz by mohl vypadat takto: úhel ve vrcholu n-úhelníka má velikost $\varphi = \pi - 2\pi/n = 2\pi(n-2)/(2n)$. Platí tedy $2n/(n-2)\varphi = 2\pi$. Abychom mohli zaplnit n-úhelníky rovinu, je třeba složit jejich vrcholy k sobě a vyplnit tím plný úhel 2π, tedy musí platit $k\varphi = 2\pi$ pro nějaké přirozené k. Tedy hodnota $2n/(n-2)$ musí být přirozené číslo. Snadno nahlédneme, že to je možné pouze pro hodnoty $n = 3, 4, 6$; pro $n > 6$ leží hodnota zlomku $2n/(n-2)$ v otevřeném intervalu (2,3).

3.6.4 Uvažme libovolnou neprázdnou množinu D dlaždic s obarvenými hranami. Vyberme z ní libovolnou dlaždici a označme její barvy jako na obr. 4.1(a) (uvedené čtyři barvy nemusí být navzájem různé, mohou být zcela libovolné). Jejím otočením o 180^o vznikne dlaždice z obr. 4.1(b). Tyto dvě dlaždice přiloženy k sobě vytvoří útvar z obr. 4.1(c), který je již periodou, neboť jím lze posouváním bez natáčení vyplnit celou rovinu (ověřte jak). Tedy připustíme-li natáčení o 180^o, každý případ problému dláždění má triviální řešení a tedy problém je rozhodnutelný — pro všechny případy dláždění roviny existuje.

3.6.5 Uvažujme množinu dlaždic D, jejichž hrany jsou obarveny barvami z neprázdné palety $\mathcal{C}$. Hledáme-li transformaci množiny D na dlaždice s obarvenými vrcholy, nejsnazší je převést informace o barvě každé hrany do barvy vrcholů s touto hranou sousedících. Každý vrchol tedy ponese informaci o původních barvách hran, jež se v něm stýkají.

U dláždění s obarvenými vrcholy navíc platí, že všechny čtyři dlaždice, které sousedí v jednom bodu svými vrcholy, musí mít tyto vrcholy stejně obarveny. Tyto čtyři shodně obarvené vrcholy musí nést informaci o barvách všech čtyř hran dlaždic, jež se v něm scházejí, viz obr. 4.2(a). Barvu vrcholu transformované dlaždice tedy budeme reprezentovat čtveřicí $\begin{pmatrix} \alpha & \\ \delta & \beta \\ & \gamma \end{pmatrix}$.



Obrázek 4.2: Převod dlaždic s obarvenými hranami na dlaždice s obarvenými vrcholy.

Uvažujme nyní, jak obarvit vrcholy dlaždice I. z obr. 4.2(b) tak, aby výsledná dlaždice mohla sousedit s transformacemi dlaždic II., III., IV. a V. tehdy a jen tehdy, když spolu sousedí původní dlaždice. Při barvení pravého horního vrcholu dlaždice I. známe barvy a a b, ale nevíme nic o barvách hran e a f sousedících dlaždic, nicméně informace o nich přesto musí být obsažena v barvě vrcholu. Východiskem proto je vytvořit množinu nových dlaždic a vystřídat na tomto jejich vrcholu všechny barvy typu $\binom{a\ s}{b}^{r}$, kde r a s nabývají všech hodnot z $\mathcal{C}$.

Když stejnou úvahu použijeme na všechny vrcholy dlaždice I., zjistíme, že je tuto dlaždici třeba transformovat na množinu dlaždic s vrcholy obarvenými barvami $\binom{a\ s}{b}^{r}$, $\binom{c\ t}{u}^{b}$, $\binom{w\ c}{v}^{d}$, $\binom{x\ a}{d}^{y}$, kde r, s, t, u, v, w, x, y nabývají všech možných hodnot z $\mathcal{C}$, viz obr. 4.2(b). Takto transformujeme každou dlaždici množiny D.

Snadno prověříme, že transformované dlaždice spolu mohou sousedit tehdy a jen tehdy, když spolu sousedí dlaždice původní, a tedy dláždění roviny transformovanou množinou dlaždic existuje tehdy a jen tehdy, existuje-li pomocí původní množiny. Kdyby byl problém dláždění pro transformovanou množinu rozhodnutelný, byl by rozhodnutelný i pro původní množinu, což je spor s nerozhodnutelností problému dláždění.

3.6.8 Pokud daný případ PKP má řešení, je uvedeno. Pokud řešení nemá, je to zdůvodněno, ovšem jsou možná i jiná zdůvodnění než která uvádíme.

(a) Případ (aab, aa), (a, baa) nemá řešení. Zdůvodnění: slova $u_{i_1} \dots u_{i_m}$, $v_{i_1} \dots v_{i_m}$ odpovídající řešení $(i_1 \dots i_m)$ by musela být shodná a tedy i obsahovat stejné počty jednotlivých symbolů. Aby to platilo pro symbol b, musí být počet jedniček v řešení $(i_1 \dots i_m)$ roven počtu dvojek. Pak ovšem nemohou být shodné počty symbolů a ve slovech $u_{i_1} \dots u_{i_m}$, $v_{i_1} \dots v_{i_m}$ a tedy řešení nemůže existovat.

(b) Případ (aab, a), (a, baa) má řešení např. (1,1,2,2).

(c) Případ (a, ba, b, ab), (aaa, b, bb, ba) nemá řešení. Důvod např.: předně index 3 se nemůže vyskytovat v žádném řešení, neboť za každý jeho výskyt by slovo $u_{i_1} \dots u_{i_m}$ obsahovalo o 1 symbol b méně než slovo $v_{i_1} \dots v_{i_m}$ a tento rozdíl nelze vyrovnat použitím ostatních indexů.

Pak tedy řešení musí začínat indexem 2 (viz tvar slov) následovaným indexem 1, čímž dostaneme situaci $u_2u_1 = baa$, $v_2v_1 = baaa$. Nyní lze použít i opakovaně index 4. Je-li použit k-krát, dostaneme slova $baa(ab)^k$, $baa(ab)^ka$, což nevede k nalezení řešení. Jediná další možnost je použít index 1, tím dostaneme slova $baa(ab)^ka$, $baa(ab)^kaaaa$ a je zřejmé, že nyní již nelze použít jiný index než 1, čímž se ovšem druhé slovo stále prodlužuje na úkor prvního. Řešení tedy neexistuje.

3.6.12 Uvažujme libovolný případ $u = (u_1, \ldots, u_n)$, $v = (v_1, \ldots, v_n)$ PKP nad jednoprvkovou abecedou. Zjevně posloupnost indexů $i_1, \ldots, i_m$ pro nějaké $m \geq 1$ je řešením tohoto případu tehdy a jen tehdy, když $|u_{i_1}| + \ldots + |u_{i_m}| = |v_{i_1}| + \ldots + |v_{i_m}|$ (dvě slova nad jednoprvkovou abecedou jsou shodná právě když mají shodnou délku). Nyní můžeme rozlišit čtyři různé možnosti:

(i) Existuje k, $1 \leq k \leq n$ takové, že $|u_k| = |v_k|$. Pak náš případ má jistě řešení, kterým je tento index k.

(ii) Pro všechna k, $1 \leq k \leq n$ platí $|u_k| > |v_k|$. Pak náš případ řešení jistě nemá.

(iii) Pro všechna k, $1 \leq k \leq n$ platí $|u_k| < |v_k|$. Pak náš případ opět nemá řešení.

(iv) Existují j, k, $1 \leq j, k \leq n$ tak, že $|u_j| > |v_j|$ a $|u_k| < |v_k|$. Pak řešením našeho případu je posloupnost

$$(\underbrace{j, \ldots, j}_{|v_k|-|u_k|}, \underbrace{k, \ldots, k}_{|u_j|-|v_j|}),$$

neboť platí (ověřte)

$$|u_j|(|v_k| - |u_k|) + |u_k|(|u_j| - |v_j|) = |v_j|(|v_k| - |u_k|) + |v_k|(|u_j| - |v_j|).$$

Libovolný případ PKP nad jednoprvkovou abecedou patří do jedné z možností (i) až (iv) a pro každou z nich lehce rozhodneme, zda existuje řešení. PKP nad jednoprvkovou abecedou je tedy rozhodnutelný.

4.2.3 Jazyk $L(G)$ je přijímán zásobníkovým automatem tehdy a jen tehdy, je-li bezkontextový. „Být bezkontextovým jazykem“ je netriviální vlastnost rekurzívně spočetných jazyků, jejíž nerozhodnutelnost plyne z Riceovy věty.

4.2.4 Pokud bychom mohli rozhodnout, zda libovolný symbol $A \in N$ je nadbytečný, mohli bychom to rozhodnout i pro počáteční neterminál S. Platí ovšem, že S je nadbytečný právě tehdy, když $L(G) = \emptyset$, a to je pro gramatiky typu 0 nerozhodnutelné (viz tabulka 4.1).

4.3.4 Každý bezkontextový jazyk lze přijímat nedeterministickým zásobníkovým automatem (ZA), který nevykonává žádné ϵ-kroky. Ukážeme, jak jej lze simulovat třístopým deterministickým lineárně ohraničeným automatem (LOA):

Na 1.stopě bude vždy vstupní slovo ZA. Na 2.stopě budeme simulovat zásobník: jestliže máme n zásobníkových symbolů a ZA v jich jednom kroku zapíše do zásobníku nejvýše k, pak LOA potřebuje n^k symbolů, které může zapisovat na 2.stopu. Víme totiž, že ZA při zpracování slova délky m udělá nejvýše m kroků a při každém zapíše do zásobníku jeden z n^k možných řetězců zásobníkových symbolů.

3. stopa slouží k odstranění nedeterminismu: nechť ZA se v každém kroku rozhoduje z max. p variant. Tyto varianty nechť odpovídají p symbolům zapisovaným na 3. stopu. Odtud plyne, že možných variant chování ZA při zpracování slova délky m je p^m a každá z nich je jednoznačně reprezentována slovem délky m na 3. stopě. Pak LOA při simulaci ZA může tyto varianty systematicky procházet (např. dle lexikografického uspořádání) a tím vyčerpat všechny možnosti chování ZA.

4.3.5 $M = (\{q_0, q_{10}, q_{11}, q_2, q_3, q_{30}, q_{31}, q_{40}, q_{41}, q_5, q_6, q_7, q_F\}, \{0,1\}, \{0,1,\$,\#,\sqcup,m\}, \delta, q_0, \{q_F\})$. Pětice náležející do δ jsou popsány níže. Nejprve nedeterministicky vybereme začátky prvního a druhého slova ω a označíme je značkou m :

$(q_0, [\$,0,1], q_0, *, \rightarrow)$ $(q_{10}, [0,1], q_{10}, *, \rightarrow)$ $(q_{11}, [0,1], q_{11}, *, \rightarrow)$
$(q_0, 0, q_{10}, m, \rightarrow)$ $(q_0, 1, q_{11}, m, \rightarrow)$ $(q_{10}, 0, q_2, m, \leftarrow)$ $(q_{11}, 1, q_2, m, \leftarrow)$

Nyní kontrolujeme, zda první kopie ω se shoduje s druhou:

$(q_2, [0,1], q_2, *, \leftarrow)$ $(q_3, 0, q_{30}, m, \rightarrow)$ $(q_{30}, [0,1], q_{30}, *, \rightarrow)$ $(q_{30}, m, q_{40}, m, \rightarrow)$
$(q_2, m, q_3, m, \rightarrow)$ $(q_3, 1, q_{31}, m, \rightarrow)$ $(q_{31}, [0,1], q_{31}, *, \rightarrow)$ $(q_{31}, m, q_{41}, m, \rightarrow)$
$(q_{40}, m, q_{40}, m, \rightarrow)$ $(q_{40}, 0, q_5, m, \leftarrow)$ $(q_5, m, q_5, m, \leftarrow)$
$(q_{41}, m, q_{41}, m, \rightarrow)$ $(q_{41}, 1, q_5, m, \leftarrow)$ $(q_5, [0,1], q_2, *, \leftarrow)$

Pokud jsme již prošli celé první slovo ω (poznáme to tak, že se oba bloky m spojily a stroj při hledání levého z nich dospěl na značku \$), zkonrolujeme, zda je zpracováno i celé druhé slovo ω :

$(q_2, \$, q_6, \$, \rightarrow)$ $(q_6, [0,1], q_6, *, \rightarrow)$ $(q_6, m, q_7, m, \rightarrow)$ $(q_7, m, q_7, m, \rightarrow)$ $(q_7, \#, q_F, \#, \downarrow)$

4.3.6 $M = (\{q_0, \ldots, q_8, q_F\}, \{1\}, \{1,\$,\#,m,\sqcup\}, \delta, q_0, \{q_F\})$. Nejprve umažeme první jedničku a označíme konec slova značkou m.

$\delta(q_0,\$,\sqcup)=(q_0,\$,\sqcup,\rightarrow)$ $\delta(q_1,1,\sqcup)=(q_1,1,\sqcup,\rightarrow)$ $\delta(q_2,1,\sqcup)=(q_3,1,m,\leftarrow)$
$\delta(q_0,1,\sqcup)=(q_1,\sqcup,\sqcup,\rightarrow)$ $\delta(q_1,\#,\sqcup)=(q_2,\#,\sqcup,\leftarrow)$

Označíme začátek slova rovněž značkou a posunujeme obě značky k sobě, až se setkají (byly-li vzdáleny od sebe o sudý počet míst):

$\delta(q_3,1,\sqcup)=(q_3,1,\sqcup,\leftarrow)$ $\delta(q_4,1,\sqcup)=(q_5,1,m,\rightarrow)$ $\delta(q_6,1,\sqcup)=(q_3,1,m,\leftarrow)$
$\delta(q_3,\sqcup,\sqcup)=(q_8,\sqcup,\sqcup,\rightarrow)$ $\delta(q_5,1,\sqcup)=(q_5,1,\sqcup,\rightarrow)$ $\delta(q_7,1,\sqcup)=(q_8,1,\sqcup,\rightarrow)$
$\delta(q_3,1,m)=(q_4,1,\sqcup,\rightarrow)$ $\delta(q_5,1,m)=(q_6,1,\sqcup,\leftarrow)$ $\delta(q_8,1,\sqcup)=(q_5,1,m,\rightarrow)$

Pokud se při srovnávání značky setkají na začátku pásky, došli jsme postupným dělením dvěma až k číslu 1. Slovo tedy mělo délku odpovídající mocnině 2 a stroj jej akceptoval. Pokud se značky setkají mimo začátek pásky, posuneme pravou značku o 1 místo vlevo a vše se opakuje pro $m := m/2$:

$$\delta(q_4,1,m)=(q_2,1,\sqcup,\leftarrow) \qquad \delta(q_8,1,m)=(q_F,1,m,\rightarrow)$$

Pokud se značky nesetkají, znamená to, že podíl získaný z m postupným dělením dvěma je lichý (a větší než jedna) a stroj slovo neakceptuje.

4.4.2 Bezkontextové gramatiky G_1 a G_2 patří rovněž do třídy kontextových gramatik, neboť vyhovují omezením kladeným na tvar bezkontextové gramatiky (viz definice např.

v [6]). Pak ovšem podle věty 4.4.1 existuje kontextová gramatika G_{12} tak, že $L(G_{12}) = L(G_1) \cap L(G_2)$.

4.5.8 Je-li R regulární, pak i $\overline{R}$ je regulární množina. Vzhledem k uzavřenosti třídy bezkontextových jazyků na průnik s regulární množinou je i $L(G) \cap \overline{R}$ bezkontextový jazyk. Přitom platí, že $L(G) \subseteq R \Leftrightarrow L(G) \cap \overline{R} = \emptyset$. Platnost poslední uvedené rovnosti je pro bezkontextové jazyky rozhodnutelná a tedy problém je rozhodnutelný.

4.5.11 Gramatiky G_U, G_V z důkazu věty 4.5.5, generující jazyky L_U, L_V odpovídající seznamům PKP, jsou lineární. Pro tyto gramatiky jsme ukázali, že je nerozhodnutelné, zda $L(G_U) \cap L(G_V) = \emptyset$.

4.5.12 Uvažujme jazyk $\overline{T_{UV}} = R_{UV} \cap S_{UV}$ z důkazu věty 4.5.10. Víme, že jazyk $\overline{T_{UV}}$ je kontextový, neboť je průnikem dvou bezkontextových jazyků, viz příklad 4.4.2. Dále bylo v důkaze věty 4.5.10 ukázáno, že jazyk $\overline{T_{UV}}$ je bezkontextový tehdy a jen tehdy, je-li prázdný, což je nerozhodnutelné.

4.5.13 Uvažujme libovolné bezkontextové gramatiky G_1 a G_2. Ke gramatice G_2 sestrojíme zásobníkový automat M tak, že $L(G_2) = L(M)$. Předpokládejme, že je rozhodnutelné, zda existuje slovo $\omega \in L(G_1)$ takové, že $\omega \notin L(M)$. Ovšem takové slovo existuje tehdy a jen tehdy, když $L(G_1) - L(M) \neq \emptyset$, tedy pokud $L(G_1) \not\subseteq L(G_2)$. Pak by tedy bylo rozhodnutelné, zda $L(G_1) \subseteq L(G_2)$, což je spor s důsledkem 4.5.9.

4.6.4

(a) $R \subseteq L$ tehdy a jen tehdy, když $R \cap \overline{L} = \emptyset$. Ovšem díky uzavřenosti třídy deterministických jazyků na doplněk a na průnik s regulární množinou podle věty 4.6.1 existuje bezkontextová $LR(k)$ gramatika G generující jazyk $R \cap \overline{L}$, a pro bezkontextové gramatiky je rozhodnutelné, zda $L(G) = \emptyset$. Tedy i původní problém, zda $R \subseteq L$, je rozhodnutelný.

(b) Jazyk $\overline{L}$ je vždy deterministický, a tedy i bezkontextový, vzhledem k uzavřenosti třídy deterministických jazyků na doplněk.

(c) Podle výsledku cvičení (b) existuje bezkontextová gramatika generující jazyk $\overline{L}$, a pro bezkontextové gramatiky je rozhodnutelné, zda generují prázdný jazyk.

(d) $L = R$ platí tehdy a jen tehdy, když $L \subseteq R$ a $R \subseteq L$. Druhé z těchto tvrzení je rozhodnutelné podle výsledku cvičení (a), první je ekvivalentní s tvrzením $L \cap \overline{R} = \emptyset$. Ovšem vzhledem k uzavřenosti třídy regulárních jazyků na doplněk a k uzavřenosti deterministických jazyků na průnik s regulární množinou podle věty 4.6.1 existuje bezkontextová $LR(k)$ gramatika G generující jazyk $L \cap \overline{R}$. Pro bezkontextové gramatiky je rozhodnutelné, zda $L(G) = \emptyset$. Tedy i první tvrzení je rozhodnutelné, a proto i původní problém, zda $L = R$.

4.6.5

(a) Jazyky L_U a L_V z důkazu věty 4.5.5 jsou deterministické, a pro tyto dva deterministické jazyky tedy platí, že je nerozhodnutelné, zda jejich průnik je prázdný.

(b) Jazyky R_{UV} a S_{UV} z důkazu věty 4.5.10 jsou deterministické, a pro tyto dva deterministické jazyky tedy platí, že je nerozhodnutelné, zda jejich průnik je bezkontextový jazyk.

(c) Uvažujme deterministické jazyky $\overline{R_{UV}}$ a $\overline{S_{UV}}$ z důkazu věty 4.5.10 a jejich sjednocení $T_{UV} = \overline{S_{UV}} \cup \overline{R_{UV}}$. Předpokládejme, že je rozhodnutelné, zda T_{UV} je deterministický jazyk. Vzhledem k uzávěrovým vlastnostem to ovšem platí tehdy a jen tehdy, když i jeho doplněk $\overline{T_{UV}}$ je deterministický (a tedy bezkontextový), což je dle důkazu věty 4.5.10 nerozhodnutelné.

(d) Jazyky L_U a L_V z důkazu věty 4.5.5 jsou deterministické, a podle věty 4.6.1 je deterministický i jazyk $\overline{L_V}$. Ovšem $L_U \subseteq \overline{L_V}$ tehdy a jen tehdy, když $L_U \cap \overline{\overline{L_V}} = \emptyset$, tedy když $L_U \cap L_V = \emptyset$, což je dle věty 4.5.5 nerozhodnutelné.

4.6.6 Podle věty 4.6.3(c) je pro dané deterministické jazyky L_1 a L_2 nerozhodnutelné, zda jazyk je $L_1 \cup L_2$ deterministický. Tento jazyk je ovšem bezkontextový a lze efektivně sestrojit bezkontextovou gramatiku, která jej generuje. Rozhodnutelnost problému, zda daná bezkontextová gramatika generuje deterministický jazyk, by tedy vedla ke sporu s tvrzením věty 4.6.3(c).

Literatura

[1] L. Adleman: Molecular computation of solutions to combinatorial problems. *Science* **266** (1994), 1021–1024.

[2] L. Brim: *Vyčíslitelnost — poznámky k přednášce.* Studijní text FI MU Brno, 2001.

[3] A. Church: An unsolvable problem of elementary number theory. *Amer. J. Math.* **58** (1936), 345–363.

[4] A. Church: A set of postulates for the foundation of logic. *Ann. Math.* **33, 34** (1933), 346–366, 839–864.

[5] J. Gruska: *Foundations of Computing.* International Thomson Computer Press, 1997.

[6] J. E. Hopcroft, J. D. Ullman: *Formálne jazyky a automaty.* Alfa Bratislava 1978.

[7] J. Kelemen: *Kybergolem.* Votobia Olomouc 2001.

[8] S.C. Kleene: General recursive functions of natural numbers. *Math. Annalen* **112** (1936), 727–742.

[9] D. C. Kozen: *Automata and Computability.* Springer-Verlag, New York, 1997.

[10] W. Pitts, W.S. McCulloch: A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* **5** (1943), 115–133.

[11] A.R. Meyer, D.M.Ritchie: The complexity of loop programs. In *Proc. ACM Natl. Meeting*, 1967, 465–469.

[12] G. Rozenberg, A. Salomaa: *Cornerstones of Undecidability.* Prentice-Hall, Englewood Cliffs, 1994.

[13] A. Salomaa, G. Rozenberg (Eds.): *Handbook of Formal Languages.* Springer-Verlag, 1997.

[14] J.C. Shepherdson, H.E. Sturgis: Computability of recursive functions. *Journal of the ACM* **10** (1963), 217–255.

[15] G. Sénizergues: L(A)=L(B)? In *Proceedings INFINITY'97*, 1–26, 1997. Electronic Notes in Computer Science 9, http://www.elsevier.nl/locate/entcs/volume9.html.

[16] C.H. Smith: *A Recursive Introduction to the Theory of Computation.* Springer-Verlag, 1994.

[17] A.M. Turing: On computable numbers, with applications to the Entscheidungsproblem. *Procs. of the London Mathematical Society* **42** (1936), 230–265.

[18] J. Wiedermann: Superturingovský výpočetní potenciál kognitivních a evolučních systémů. Technická zpráva 832, únor 2001, ÚIVT ČSAV, http://www.uivt.cas.cz.