VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

DEMONSTRACE SKÁKAJÍCÍCH AUTOMATŮ

DEMONSTRATION OF JUMPING AUTOMATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **LADISLAV RŮŽIČKA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. ZBYNĚK KŘIVKA, Ph.D.**
SUPERVISOR

BRNO 2017

Abstrakt

Tato práce se zabývá demonstrací nově zkoumaného výpočetního modelu pro popis formálních jazyků, a to skákajícího automatu. Místo souvislého čtení vstupního řetězce, jak je tomu u konvenčních konečných automatů, tak u skákajícího automatu je proveden skok přes nějaké symboly, a poté je přečten symbol. V této práci se zejména budeme zabývat hledáním praktického algoritmu pro určení problému členství vstupního řetězce do jazyka popsaného skákajícím automatem. Ukážeme, že problém členství může být redukován na problém hledání nějakého nezáporného celočíselného řešení pro formuli v Presburgové aritmetice bez kvantifikátorů. Z této formule jsme schopni jednoznačně definovat jazyk přijímaný skákajícím automatem. Najdeme podmnožinu takových skákajících automatů, pro které lze vyřešit problém členství v polynomiálním čase. Zmíníme se také, že předchozí formule lze převést na konečný automat s více čtecími hlavami. Bohužel pro problém členství obecného skákajícího automatu hledání nezáporné číselného řešení je nedostačující, nicméně metoda může zmenšit prohledávaný stavový prostor. Uvedeme další možné heuristiky, které výrazně urychlují výpočet problému členství pro obecné skákající automaty.

Abstract

This paper is concerned with demonstration of newly investigated jumping finite automata. Unlike conventional finite automata that read input words continuously these automatas make a jump over some symbols and from there it can read a symbol. In this paper we will be mostly focused on finding a practical algorithm for solving the membership problem. As will be shown the membership problem for jumping finite automata can be reduced to finding a non-negative integral solution to a Quantifier-Free Presburger arithmetics formula. From such formula we are able to determine whole infinite language of jumping finite automata. We will show that some subset of jumping finite automata can be solved in polynomial time. We note that formula in Presburger arithmetics can be transformed to the corresponding concurent finite automata. Unfortunately for general jumping automata finding non-negative solution is not sufficent but it can reduce search space. Other heuristics will be presented that increase the effectivity for the general jumping finite automata acceptance process.

Klíčová slova

skákající konečný automat, obecný skákající automat, problém členství, celočíselné linearní programování, podmínkové celočíselné programování, zpětné navrácení, heuristiky

Keywords

jumping finite automata, general jumping finite automata, membership problem, integer linear programming, constraint integer programming, backtrack, heuristics

Citace

RŮŽIČKA, Ladislav. *Demonstrace skákajících automatů*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Křivka Zbyněk.

Demonstrace skákajících automatů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Zbyňka Křivky. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Ladislav Růžička
17. května 2017

Poděkování

Rád bych poděkoval Ing. Zbyňkovi Křivkovi Ph.D. za cenné rady při zpracování této práce, věcné připomínky a vstřícnost při konzultacích.

Obsah

Kapitola 1

Úvod

Úvod je rozdělen na čtyři sekce následovně. V sekci 1.1 definujeme pojem výpočetní model a krátce se budeme zabývat tradičními výpočetními modely pro popis formálních jazyků. V sekci 1.2 se zmíníme o některých nekonvenčních modelech. Zejména nás pak budou zajímat modely pro popis formálních jazyků, a to především skákající automaty a gramatiky. Sekce 1.3 popisuje hlavní cíle této technické zprávy. V sekci 1.4 je uvedena organizace obsahu technické zprávy.

1.1 Výpočetní model

S pojmem výpočetní model se poprvé setkáváme ve třicátých letech dvacátého století, kdy Alan Turing a Alonzo Church představili formální teoretické výpočetní modely Turingova stroje [25] a lambda kalkulu [3]. Důvodem vzniku těchto matematických modelů byla otázka, jestli existuje algoritmus, který dokáže určit řešitelnost libovolného matematického tvrzení definovaného pomocí predikátové logiky prvního řádu. Entscheidungsproblem neboli rozhodovací problém položil roku 1928 David Hilbert [10]. Allan Turing a Alonzo Church nezávisle na sobě dokázali, že tento rozhodovací problém nelze vyřešit žádným algoritmem respektive žádným výpočetním modelem. Přesto tyto modely určily směr vývoje digitálních počítačů. Turingův stroj se stal předlohou pro stroj s náhodným přístupem do paměti, který navíc modeluje fyzicky omezené limity paměti. Lambda kalkul se stal základem pro funkcionální programovací jazyky.

Výpočetní model je abstraktní specifikací toho, jak může probíhat výpočet nějakého problému [6]. Obsahuje množinu operací a jejich cenu pro vykonání. Hlavním cílem pro výpočetní model je snaha zobecnit koncept algoritmu bez toho, aniž by odkazoval na konkrétní programovací jazyk nebo na fyzické zařízení. Jinými slovy výpočetní model abstrahuje od zařízení, na kterém je prováděn výpočet. Churchova-Turingova teze tvrdí, že ke každému spočetnému algoritmu existuje ekvivalentní Turingův stroj a, že každý možný výpočet lze úspěšně uskutečnit algoritmem běžícím na počítači, je-li k dispozici dostatek času a paměti. Například výpočet Turingova stroje probíhá tak, že z neomezené pásky jsou postupně čteny jednotlivé symboly. Na základě vnitřního stavu Turingova stroje a právě čteného symbolu se změní vnitřní stav podle přechodové tabulky, na pásku se zapíše nový symbol a páska respektive čtecí hlava se posune o jednu pozici doleva nebo doprava. V teoretickém modelu je čtecí páska neomezená, avšak v každém výpočetním kroku je zaplněno konečně mnoho políček, reprezentující symboly. Základem Turingova stroje je procesorová jednotka, tvořená konečným automatem, program ve tvaru přechodové tabulky a neomezená páska.

Hlavním zájmem pro analyzování míry složitosti algoritmů, určování fyzických limitů a studování dalších vlastností výpočetních modelů se stala skupina automatů, mezi které patří i Turingův stroj s největší vyjadřovací schopností. Od Turingova stroje se odvozují třídy složitosti P a NP pro popis určitých rozhodovacích problémů. Třída složitosti P obsahuje všechny problémy řešitelné pomocí deterministického Turingova stroje v polynomiálním čase. Naproti tomu NP obsahuje všechny rozhodovací úlohy řešitelné v nedeterministickém Turingově stroji v polynomiálním čase, který umožňuje výpočet rozvětvit a všechny rozvětvené výpočty provádět současně.

Ke každému automatu můžeme přiřadit formální jazyk, který je vlastně množinou všech sekvencí přechodů do koncového stavu. Konečný automat má nejnižší vyjadřovací sílu podle Chomského hierarchie, avšak mají důležitou roli v aplikované informatice. Jazyku přijímaným konečným automatem se říká regulární jazyk. Nejčastější využití nejen konečných automatů, ale i ostatních modelů, je určení členství řetězce do jazyka popsaného konečných automatem, jinými slovy jestli daný konečný automat přijímá vstupní řetězec. Příkladem použití konečných automatů může být lexikální analýza, zpracovávání textu,...

1.2 Nekonvenční výpočetní modely

Nový výpočetní model, nebo nový prvek v tradičním modelu, obvykle vytvoří novou rodinu formálních jazyků a nové paradigmata v procesu vývoje softwaru. V posledních desetiletích vznikalo několik nových teoretických výpočetních modelů. Nejznámější jsou kvantové výpočetní modely založené na kvantové mechanice [8] a biologií inspirované výpočetní modely jako například celulární automaty, neuronové sítě atd. Nové teoretické výpočetní modely jsou často objektem ke zkoumaní a studiu.

Další nově zkoumanou skupinou výpočetních modelů jsou modely založené na nespojitém zpracovávání informací. Většina dosud zkoumaných modelů jako např. skupina automatů jsou založeny na spojitém zpracovávání informace. Čtení probíhá striktně spojitým posunem čtecí hlavy zleva doprava. Nicméně moderní výpočetní metody jsou založeny na nespojitém zpracovávání informací. Samotný výpočetní krok může být proveden kdekoliv uvnitř informace, a tím pádem může přeskočit velkou část informace.

Snahou bylo zavedení konceptu nespojitého zpracování informace do již existujících klasických formálních modelů. Výsledkem byl výpočetní formální model nazvaný skákající konečný automat[17] a jejich generující modely skákající gramatiky[12].

Hlavní myšlenkou skákajících automatů je, že čtení vstupního řetězce neprobíhá striktně zleva doprava, ale před přečtením slova, skákající automat může provést skok na libovolné místo ve vstupním řetězci. Se zavedením skákajícího automatu nastávají následující praktické otázky. Jakým způsobem lze zjistit, jestli daný řetězec je přijímán skákajícím automatem? Lze jasným způsobem charakterizovat jazyk definovaným skákajícím automatem? Algoritmus pro určení členství řetězce do jazyka popsaného skákajícím automatem je problematický z několika důvodů. Může existovat situace, kdy v řetězci existuje více možností přečtení slova. Pokud bude ve vstupním řetězci existovat zároveň více stejných symbolů, nutných pro přechod do dalšího stavu, bude to mít za následek, že bude existovat i více způsobu, v jakém pořadí budou tyto symboly přečtena.

Konečným automatům lze přesně charakterizovat jejich přijímaný jazyk pomocí regulárních výrazů. U skákajících automatů může být problematické nalézt takovou charakteristiku jejich přijímaného jazyka.

1.3 Cíl

Hlavním cílem technické zprávy je seznámit se s nekonvenčním modelem skákajícího automatu pro popis formálního jazyka a navrhnout efektivní algoritmus pro určení členství vstupního řetězce do jazyka popsaného skákajícím automatem. Mimo jiné se pokusíme matematicky charakterizovat jazyk popsaný skákajícím automatem.

1.4 Organizace textu

Kapitola 2 zavádí použitou notaci pro tuto technickou zprávu. V kapitole 3 definujeme skákající automat a klasifikujeme si jej podle nejdelší délky aplikovaného pravidla na obecný skákající automat a jednoduchý skákající automat. Také definujeme generující modely skákajících gramatik a další vlastnosti. Kapitola 4 se zabývá algoritmy pro určování členství libovolného řetězce do jazyka popsaného jednoduchým skákajícím automatem. Popíšeme si, jak lze převést rozhodovací problém na problém o systému lineárních rovnic pro přirozená čísla respektive formuli v Presburgerově aritmetice. A následně matematicky přesně charakterizujeme pro libovolný jednoduchý skákající automat jeho přijímaný jazyk. Kapitola 5 se zabývá možnými způsoby řešení diofantických lineárních rovnic a formule Presburgerovy aritmetiky. Na základě výpočtu popíšeme vztahy mezi skákajícími automaty a lineárním diofantickým systémem. Ukážeme, že existuje podmnožina skákajících automatů, které lze vyřešit v polynomiálním čase. Zmíníme se o převodu Presburgerově aritmetiky na konečný automat s více čtecími hlavami. V kapitole 6 se řeší přijímaní obecného skákajícího automatu, pro který nestačí jenom vyřešit lineární diofantický systém, ale je nutné i prohledat stavový prostor. Zjistíme, že při přijímaní dochází často k prohledávání již vyřešeného podproblému a zavedeme několik heuristik. V kapitole 7 popisujeme postup implementace algoritmu a reprezentujeme některé výsledky. Kapitola 8 shrnuje celou práci a dosažené výsledky.

Kapitola 2

Prerekvizity

Práce předpokládá obecnou znalost konečných automatů. Všechna použitá notace v této práci pro popis formálních a matematických problému bude zde uvedena.

Pro množinu vstupní abecedy Σ, Σ^* reprezentuje volný monoid generovaný množinou Σ s operací konkatenace. Nechť je definován řetězec $w \in \Sigma^*$, potom $|w|_a$ označuje počet výskytů symbolu a v řetězci w a $|w|$ značí počet všech symbolů v řetězci w. Jednotlivé symboly v řetězci označujeme jako $w_1 w_2 ... w_n$, kde $n = |w|$. Množina všech permutací řetězce w je definováno jako $perm(w)$. Pro množinu Σ, $|\Sigma|$ značí kardinalitu množiny. Konečný automat M je pětice (Q, Σ, R, s, F), kde Q je konečná množina stavů, s je počáteční stav, F je množina koncových stavů a R je konečná množina pravidel $Q \times \Sigma^* \times Q$. Pravidlo $(p, y, q) \in R$ píšeme ve tvaru $py \to q$. Pokud $|y| = 0$, symbol značíme jako $y = \varepsilon$, potom pravidlo $py \to q$ znamená, že jsme se ze stavu p dostali do stavu q bez toho, aniž bychom přečetli symbol. Nechť $x, y \in \Sigma^*$ a $py \to q \in R$, potom konečný automat provede derivační krok z pyx na qx, značeno jako $pyx \Rightarrow qx$. Tranzitivní uzávěr je definován jako $\Rightarrow^*$. Jazyk konečného automatu je definován jako $L(M) = \{w \in \Sigma^* : sw \to^* f, f \in F\}$.

Pokud pro všechny pravidla $py \to q \in R$ platí $|y| = 1$, potom konečný automat je bez ε přechodů. Konečný automat je deterministický, pokud pro všechny pravidla $py \to q \in R$ platí $|y| \geq 1$ a pro každý $p \in Q$ a každý $a \in \Sigma$ neexistuje další stav q takový, že $pa \to q \in R$. Každý nedeterministický konečný automat lze převést na deterministický tak, že $L(M) = L(M')$. Nechť je definován $\varepsilon - uzávěr(p) = \{q : q \in Q, p \Rightarrow^* q\}$, potom převod na konečný automat bez ε přechodů $M' = (Q, \Sigma, R', s, F')$ je definován následovně. Pro každý stav $p \in Q$ proveď

$$R' = R' \cup \{pa \to q : p'a \to q \in R, a \in \Sigma, p' \in \varepsilon - uzávěr(p), q \in Q\}$$

a potom množina koncových stavů bude $F' = \{p : \varepsilon - uzávěr(p) \cap F \neq \emptyset\}$. Převod konečného automatu bez ε přechodů na deterministický konečný automat $M' = (Q', \Sigma, R', s, F')$ je definován následovně. Nechť množina stavů $Q' = \{s\}$, potom pro každý stav $q \in Q'$ ve tvaru $q = (q_1, q_2, ..., q_n)$, kde $q_1, q_2, ..., q_n \in Q$ a symbol $a \in \Sigma$ udělej

$$R' = R' \cup \{qa \to p \in R\} \text{ a } Q' = Q' \cup p,$$

kde stav $p = (q' : q_i \in q, q_i a \to q', i = 1, 2, ...n)$ a množina koncových stavů $F' = \{q : q \in Q', q_i \in F, i = 1, 2, ...n\}$. Pro každý deterministický konečný automat M existuje unikátní minimální deterministický konečný automat M' takový, že $L(M) = L(M')$. Deterministický konečný automat je minimalní, pokud neobsahuje neukončující a nerozlišitelné stavy. Stav q je ukončující, pokud existuje řetězec $w \in \Sigma^*$, pro který platí $qw \Rightarrow^* f$, $f \in F$, jinak je neukončující. Odstranění nerozlišitelných stavů je popsáno následovně.

Nechť $P = \{Q - F, F\}$, dokud pro každou podmnožinu $B \in P$ existuje $a \in \Sigma$ takové, že $q_1 a \to q_1' \in R$ a $q_2 a \to q_2' \in R$ pro $q_1, q_2 \in B$, $B' \in P$, $q_1' \in B'$ a $q_2' \notin B'$, rozděl B následovně

$$P = \{P - B \cup B_1, B_2\},$$

kde $B_1 = \{q : q \in B, qa \to q' \in R,\ q' \in B'\}$ a $B_2 = B - B_1$.

Reverzace konečného automatu M je definována jako $rev(M) = (Q, \Sigma, s', R', F')$, kde množina pravidel R' je definována jako

$$R' = \{qy \to p :\ py \to q \in R\} \cup \{s'\varepsilon \to f\ f \in F\}$$

a množina koncových stavů $F' = \{s\}$. Průnik dvou konečných automatů L a K značíme jako $L \cap K$. Pro $p_1, p_2 \in L$ a $q_1, q_2 \in K$, množina pravidel průniku dvou automatu je $(p_1, q_1)a \to (p_2, q_2)$ pokud $p_1 a \to p_2$ a $q_1 a \to q_2$. Sjednocení konečných automatů značíme jako $L \cup K$.

Nechť množina $\mathbb{N}$ reprezentuje množinu všech nezáporných číslech $\mathbb{N} = 0, 1, 2..., \infty$.

$Ax = C$ reprezentuje systém o m lineárních rovnicích a o n neznámých, kde A je matice koeficientů, x je vektor neznámých a C je pravá strana rovnice. Maticově zapsáno následovně:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix}$$

Pokud všechny prvky C jsou nulové, pak o systému říkáme, že je homogenní, jinak je heterogenní. A^T značí transponovanou matici. Pokud $m > n$, pak je systém naddeterminovaný, $m = n$ determinovaný a $m < n$ poddeterminovaný. Determinant matice je značen jako $det(A)$. Jednotková matice $n \times n$ je čtvercová matice, která má na hlavní diagonále jedničky a nuly na ostatních. Horní trojúhelníkový tvar má samé nuly pod hlavní diagonálou $A_{ij} = 0$ pro $i > j$. Unimodulární matice je čtvercová matice $n \times n$, kde $det(U) = \{-1, 1\}$. Pro neznámou x, $\lfloor x \rfloor$ znamená zaokrouhlení na celé číslo směrem k nekonečnům.

Nechť množina B obsahuje lineárně nezávislé vektory $b_1, ..., b_m$, pak se bodovou mříží z B rozumí množina $L = \{\sum_{i=1}^{m} a_i b_i : a_i \in Z\}$.

$(a_1 x_1 + a_2 x_2 + ... + a_n x_n - c = 0)\ mod\ d$ značí všechny vektory x takové, že $x_i \in < 0, d)$ a zároveň $a_1 x_2 + a_2 x_2 + ... a_n x_n - c \% d = 0$

Nechť je definován okruh polynomů pro obor čísel K, značeno $K[x]$, jako množina všech polynomů ve tvaru $a_0 + a_1 x + ... + a_n x^n$, pro nějaké nezáporné celočíselné n a pro všechny $a_0, ..., a_n \in K$. Ideál I je podmnožina prvků v okruhu $K[x]$, která má vlastnost, když $x \in K$ a $y \in I$, potom xy a yx patří do I.

Kapitola 3

Skákající automaty a gramatiky

V této kapitole si formálně definujeme skákající automaty a popíšeme si, jak probíhá derivační krok respektive skok. Ukážeme si, že proces určení členství řetězce do jazyka popsaného skákajícím automatem závisí na délce aplikovaného pravidla respektive počtu odstraněných symbolů z řetězce. V závislosti na délce aplikovaného pravidla si rozdělíme skákající automat na obecný skákající automat a jednoduchý skákající automat. Zmíníme se o hlavních problémech, které mohou nastat při zjišťování problému členství řetězce pro oba typy skákajících automatů. S analogií konečných automatů zavedeme pro skákající automaty generující model skákající gramatiky. A v neposlední řadě porovnáme vyjadřovací sílu jednoduchých a obecných skákajících automatů vůči skákající gramatice.

3.1 Skákající automat

Skákající automat je definován stejně jako jako konečný automat s jediným rozdílem a to, že čtení řetězce neprobíhá postupným posunem čtecí hlavice zleva doprava, ale skokem na libovolnou pozici v čteném řetězci.

Definice 1 *Skákající automat [17] je definován jako pětice*

$$J = (Q, \Sigma, R, s, F),$$

kde Q reprezentuje konečnou množinu stavů, Σ je vstupní abeceda, $R \subseteq Q \times \Sigma^* \times Q$ představuje konečnou množinu pravidel. Místo relačního zápisu pravidla $(p, y, q) \in R$, zapisujeme ve tvaru $py \to q \in R$. $F \subseteq Q$ značí množinu koncových stavů, s je počáteční stav. Binární relace skok, symbolicky zapsána jako $\curvearrowright$, pro konfiguraci $\Sigma^* Q \Sigma^*$, je definována následovně. Nechť pro libovolné řetězce $x, z, z', x' \in \Sigma^*$ platí $xz = x'z'$ a zároveň existuje pravidlo $py \curvearrowright q \in R$, potom J provede skok z $xpyz$ do $x'qz'$. Symbolicky zapsáno jako

$$xpyz \curvearrowright x'qz'.$$

S analogií konečného automatu, sekvenci skoků značíme jako $\curvearrowright^m$, kde $m \geq 0$. Nechť $\curvearrowright^*$ představuje tranzitivní-reflexivní uzávěr, potom jazyk přijímaný J,zapsáno jako $L(J)$, je definovaný následovně

$$L(J) = \{uv : \ u, v \in \Sigma^*, usv \curvearrowright^* f, f \in F\}.$$

Předtím než se budeme zabývat otázkou přijímaní, definujeme si některé vlastnosti skákajících automatů.

Definice 2 *Nechť $J = (Q, \Sigma, R, s, F)$ je skákající automat. Graf přechodů $G = (Q, R)$ vzniklý z J je orientovaný multigraf, kde každé $py \to q \in R$ je interpretováno jako hrana ze stavu $p \in Q$ do stavu $q \in Q$ oceněná řetězcem y. Cesta je posloupnost stavů $q_1, q_2, ..., q_n$, $n \geq 2$, a pro kterou platí, že v G existuje hrana z daného vrcholu do jeho následníka $q_i y \to q_{i+1} \in R$ pro $i = 1, 2, ..., n$. Stav $q \in Q$ je dosažitelný, pokud existuje cesta z s do q v G a zapisujeme jako $py_1y_2...y_n \rightsquigarrow q$.*

Teorém 3 *Nechť je zadán skákající automat $J = (Q, \Sigma, R, s, F)$ s ε přechody. J lze převést na ekvivalentní skákající automat J' bez ε přechodů tak, že $L(J) = L(J')$.*

Důkaz 4 *Převod je proveden s analogií převodu konečného automatu s ε přechody na konečný automat bez ε přechodů.*

Teorém 5 *Nechť je zadán skákající automat M, pak M lze převést na ekvivalentní deterministický skákající automat tak, že $L(M) = L(M')$.*

Důkaz 6 *Převod je proveden s analogií převodu konečného automatu bez ε přechody na deterministický.*

Ve skutečnosti deterministický skákající automat nebude v pravém slova smyslu deterministický, protože pokud z jednoho stavu existuje více než jedno pravidlo, potom v nějakých případech bude možné aplikovat více pravidel současně. Nastává otázka, jestli skákající automat bude deterministický, pokud bude mít z každého stavu maximálně jedno pravidlo. Odpověď zní ano, ale jen v určitém případu, o kterém se budeme bavit později. Takový skákající automat pak bude obsahovat jenom jednu cestu.

Teorém 7 *Nechť je zadán deterministický skákající automat J. J lze převést na minimalistický skákající automat J' tak, že $L(J) = L(J')$.*

Důkaz 8 *Převod je proveden s analogií převodu konečného deterministického automatu na minimalistický.*

Všechny převody konečných automatů jsou uvedené v kapitole 2.

Nyní se vraťme zpátky k otázce členství řetězce do jazyka popsaného skákajícím automatem. Při čtení řetězce se potřebné symboly pro přechod do jiného stavu mohou vyskytovat kdekoliv v řetězci. Navíc pokud v řetězci bude existovat více stejných symbolů, nutných pro přechod, pak může záležet i na tom, v jakém pořadí budou čteny. Pokud bude pro daný stav možné aplikovat více různých přechodů, pak nastává další otázka, který přechod aplikovat jako první. I kdybychom nedeterministický skákající automat převedli na deterministický, pak pořád nebudeme vědět, které pravidlo aplikovat jako první, protože dvě různé sekvence symbolů se mohou zároveň nacházet kdekoliv v řetězci. Při rozhodování, jestli daný řetězec je přijímán skákajícím automatem, tato skutečnost přidává prvek nedeterminismu, který nelze tak jednoduše odstranit.

Zkusme si doposud získané skutečnosti ukázat na příkladu.

Příklad 9 *Nechť je definován řetězec $w = bababaa$ a skákající automat*

$$M = (\{s, q, f\}, \{a, b\}, \{sa \to q, sb \to q, qba \to f, fbab \to s\}$$

Z počátečního stavu s se můžeme dostat do stavu q přečtením buď symbolu a nebo b. Protože a i b se nachází v řetězci w, musíme si zvolit, které pravidlo aplikujeme jako první. Zvolíme si tedy první možnost. Symbol a lze přečíst na čtyřech pozicích jmenovitě $2, 3, 5$ a 6. Budeme postupně zkoušet číst symboly v pořadí jejich pozic. Odstraníme symbol a z pozice 2. Nově vzniklý řetězec je $w' = bbabaa$. Dalším pravidlem by bylo odstranění symbolů ba, ale protože odstraněním jakéhokoliv ba v w' nezískáme bab, musíme se vrátit do počátečního stavu s a vložit zpátky a na pozici 2. Stejný postup zopakujeme, ale tentokrát odstraníme a na pozici w_3 a dostaneme $w' = babbaa$. Odstraníme ba na pozici 4, abychom v dalším kroku mohli přečíst bab a poté a. Jsme v koncovém stavu a zároveň $w' = \varepsilon$, tudíž platí $w \in L(M)$ a problém členství je vyřešen.

Je zřejmé, že při rozhodování členství řetězce do jazyka popsaného skákajícím automatem může záležet na pořadí čtených symbolů. Položme si otázku, kdy bude nebo nebude záležet na pořadí čtených symbolů. V předchozím příkladu jsou aplikovány pravidla různé délky.

Záležet bude jenom tehdy, pokud při odstranění nějakých symbolů vznikne nový způsob, jak pokračovat v procházení automatu. Taková situace vznikne jenom tehdy, když ve skákajícím automatu existuje alespoň jedno pravidlo, při kterém přečteme sekvenci symbolů. Hlavním důvodem je, že pravidla s větším počtem symbolů než jedna se mohou překrývat. Například pokud v řetězci odstraníme uprostřed řetězce jeden symbol, potom může vzniknout jiná variace sekvence symbolů, která může být dále odstraněna aplikováním pravidla. Kdybychom přečetli stejný symbol někde jinde v řetězci, potom bychom nedostali potřebnou variaci symbolů pro další přechod. Naproti tomu když jsou všechny pravidla délky jedna, potom nemůže vzniknout žádné překrývání. Problém přijímaní řetězce skákajícím automatem můžeme rozdělit podle největší délky aplikovaného pravidla na dvě části.

Definice 10 *Nechť je zadán skákající automat $J = (Q, \Sigma, R, s, F)$. Pokud pro každé pravidlo $py \to q \in R$, platí $|y| \leq 1$, potom J označujeme jako* jednoduchý skákající automat *(ve zkratce napsáno jako JFA) a y reprezentuje symbol. Jinak J je* obecný skákající automat *(ve zkratce GJFA) a o y hovoříme jako o slovu.*

V práci [17] je skákající automat speciální případ obecného skákající automatu, u kterého říkáme, že má stupeň $n = 1$. V této práci termín skákající automat představuje množinu všech skákající automatů tj. jednoduchých a obecných skákajících automatů. Pokud budeme mluvit o obecném skákajícím automatu, pak máme na mysli automat, ve kterém existuje alespoň jedno pravidlo s délkou slova $|y| > 1$.

3.2 Jednoduchý skákající automat

Uvažujme nad jednoduchým skákajícím automatem. V každém kroku odstraňujeme kdekoliv z řetězce právě jeden symbol. To znamená, že odstraněním jednoho symbolu nemůže vzniknout jiný symbol než, který je již obsažen v původním řetězci. Tudíž při odstranění jednoho symbolu nezáleží na tom, kde ho v řetězci odstraníme.

Teorém 11 *Nechť je definován JFA $J = (Q, \Sigma, R, s, F)$ a $w \in L(J)$. Protože řetězec $w \in L(J)$, potom musí existovat cesta taková, že $sy_1y_2...y_n \rightsquigarrow f$, $f \in F$, pro nějaké $n \geq 2$.. Nechť $y_i = y_j = a$ a nechť $w = w'aw''a$, potom nezáleží na tom, které a odstraníme jako první.*

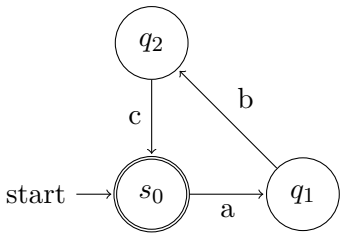
Důkaz 12 *JFA může odstranit v každém kroku jenom jeden symbol. Vzhledem k cestě jsme museli aplikovat nějaké pravidla ve tvaru* $qa \to p \in R$. *Pro symbol* $a \in \Sigma$ *jsme museli přesně* $|w|_a$ *krát použít nějaké takové pravidlo. Protože* $w \in L(J)$, *potom v koncovém stavu musí být odstraněny všechny jednotlivé symboly. Tudíž nezáleží na pořadí čtení symbolů.*

Vraťme se k předchozí úvaze o deterministickém skákajícím automatu.

Definice 13 *Nechť je zadán JFA* $J = (Q, \Sigma, R, s, F)$. J *je úplně deterministický, pokud* J *je deterministický a zároveň z každého stavu* $q \in Q$ *existuje maximálně jedno pravidlo* $qy \to p \in R$.

Demonstrujeme na příkladu.

Příklad 14 *Nechť je definován řetězec* $w = acbcabacb$ *a úplně deterministický JFA* $J = (Q, \Sigma, R, s, F)$ *následovně.*



Obrázek 3.1: Jednoduchý skákající automat z [17]

Z počátečního stavu automat postupně z w *odstraňuje symboly a,b,c. Tyto kroky se neustále opakují dokud, není přečten celý řetězec nebo dokud existují potřebné symboly pro přechod do dalšího stavu. Po každé se do koncového stavu dostaneme po přečtení právě třech symbolů, proto pro* $w \in L(M)$ *musí platit* $|w| = 3x$,$x \in \mathbb{N}$. *Navíc platí, že počet symbolů* a, b, c *se musí rovnat, tudíž jazyk* $L(J)$ *je definován matematicky*

$$L(J) = \{x \in \Sigma^* : \ |x|_a = |x|_b = |x|_c\}.$$

Pro řetězec w *platí* $w \in L(J)$, *protože* $|w|_a = |w|_b = |w|_c$. *Pro tento jednoduchý skákající automat tedy nezáleží na pořadí čtených symbolů a jediným kritériem je počet výskytu symbolů. Všechny permutace řetězce* w *jsou přijímány, protože je pokaždé dodržena předchozí podmínka. Všimněte si, že tento jednoduchý skákající automat lze přímým způsobem převést na tři rovnice o jedné neznámé.*

Teorém 15 *Nechť je dán jednoduchý skákající automat* $J = (Q, \Sigma, R, s, F)$ *a řetězec* $w \in \Sigma^*$ *pro který, platí* $w \in L(J)$. *Potom pro řetězec* w *bude platit* $perm(w) \in L(J)$.

Důkaz 16 *Je zřejmé, že pokud* $w \in L(J)$, *pak* w *musel vzniknout cestou z koncového stavu do počátečního postupným přidáváním symbolů. Aplikujeme operaci zrcadlení(reverzaci) na* J *a místo odebíraní symbolů, budeme symboly vkládat na libovolnou pozici v řetězci. Nechť je zadán prázdný řetězec* $w' = \varepsilon$. *V prvním kroku vkládáme symbol do prázdného řetězce. V dalším kroku máme dvě možnosti, kam vložit symbol. Obecně pro každý následující krok bude existovat* $n + 1$ *možností, kde* n *je aktuální délka řetězce. Pro sekvenci přechodů z* w' *do* w *bude existovat celkem* $(0 + 1)(1 + 1)...(|w| - 1 + 1) = |w|!$ *možností, jak generovat řetězec. Protože* $rev(J)$ *generuje všechny permutace řetězce* w, *potom máme další důkaz, že nezáleží na pořadí čtení symbolů.*

Ve skutečnosti existuje méně permutací řetězce, protože některé symboly se opakují. Tím pádem vzniknou opakující se řetězce. Počet všech neopakujících se permutací řetězce se spočítá jako permutace s opakováním.

Lemma 17 *Nechť je dán JFA $J = (Q, \Sigma, R, s, F)$ a řetězec $w \in L(J)$. Celkový počet přijímaných unikátních řetězců vzhledem k w je* $\frac{|w|!}{\prod_{a \in \Sigma} |w|_a!}$.

3.3 Skákající gramatika

Je zřejmé, že tento model, vzniklý rezervací jednoduchého skákajícího automatu, generuje všechny řetězce jazyka $L(J)$. Takový vkládací výpočetní model lze také charakterizovat jako skákající gramatiku. Skákající gramatika je založena na konečné množině gramatických pravidel, které generují řetězce daného jazyka.

Definice 18 *Obecná skákající gramatika [12] je čtveřice $G = (V, T, P, S)$, kde V je abeceda terminálů a neterminálů, $T \subset V$ je množina terminálů. Nechť $N = V - T$ značí množinu neterminálů, potom P je relace z N^* do $V*$, a $SV - T$ je počáteční neterminál. Místo relačního zápisu pravidla $(x, y) \in P$, píšeme symbolicky jako $x \to y \in P$. Nechť je zadána relace $r \in V^*$, potom r^n označuje sekvenci derivačních kroků. Nechť r^* představuje tranzitivní-reflexivní uzávěr. Jazyk generující G je definován jako*

$$L(G, r) = \{x \in T^* : Sr^*x\}.$$

Derivační krok pro skákající gramatiku G a pro $u, v \in V^*$ může být definován následujícími způsoby.

(i) $u \ {}_{rj}\!\!\Longrightarrow v$, pokud existuje $x \to y \in P$ a w, t, z takové že, $u = wtxz$ a $v = wtxz$

(ii) $u \ {}_{lj}\!\!\Longrightarrow v$, pokud existuje $x \to y \in P$ a w, t, z takové že, $u = wtxz$ a $v = wtxz$

(iii) $u \ {}_{j}\!\!\Longrightarrow v$, pokud $u \ {}_{lj}\!\!\Longrightarrow v$ nebo $u \ {}_{rj}\!\!\Longrightarrow v$

Definice 19 *G je bezkontextová gramatika pokud, pro každé $x \to y \in P$ platí $x \in (V - T)$.*

Definice 20 *G je lineárně-pravá gramatika, pokud G je bezkontextová gramatika a pro každé $x \to y \in P$ platí $y \in T^*(V - T) \cup T^*$.*

Definice 21 *G je regulární gramatika, pokud G je bezkontextová gramatika a pro každé $x \to y \in P$ platí $y \in T(V - T) \cup T$.*

Teorém 22 *Každý jednoduchý skákající automat lze převést na regulární gramatiku.*

Důkaz 23 *Jednoduchý skákající automat lze převést lineárně-levou skákající gramatiku s analogií reverzace jednoduché skákajícího automatu. Nechť je dán JFA $J = (Q, \Sigma, R, s, F)$. Stavy J reprezentují neterminály. Počáteční stav s bude novým počátečním neterminálem v G. Vstupní symboly Σ jsou terminální symboly G. Při generování řetězce je přítomen právě jeden neterminální symbol dokud poslední derivační krok nevytvoří poslední terminální symbol. Stejně můžeme převést jednoduchý skákající automat na lineárně-pravou skákající gramatiku. Tudíž JFA lze převést na regulární gramatiku.*

Protože jednoduchý skákající automat můžeme převést na lineárně-pravou skákající gramatiku, potom přijímaný řetězec JFA nemusíme generovat vzhledem ke zpáteční cestě. Generování řetězců přijímaných JFA může probíhat vkládáním symbolů do prázdného řetězce z počátečního stavu do koncového stavu.

3.4 Obecný skákající automat

Nyní se vrátíme k obecným skákajícím automatům. Je zřejmé, že bude záležet na pořadí čtení slov, protože odstranění slova v jakémkoliv kroku výpočtu může vytvořit jiné slovo potřebné pro přechod. Prvním prvkem nedeterminismu je, že nevíme, v jakém pořadí máme slova přečíst. Pokud v daném stavu existuje více možností, jak přejít do jiného stavu, pak ani nedokážeme určit, jaké pravidlo si vybrat jako první.

Teorém 24 *Každý obecný skákající automat lze převést na lineárně-levou skákající gramatiku, ale nelze jej převést na regulární.*

Důkaz 25 *Zkusme převést obecný skákající automat do obecné skákající gramatiky a zkoumat jazyk přijímaný obecným skákajícím automatem. V prvním kroku budeme mít zase jednu možnost kam vložit slovo. Ale v dalších krocích bude počet možností záležet na délce vkládaného slova. Například když budeme vkládat slova v pořadí* $y_1 = abb, y_2 = a, y_3 = bb$*, pak v prvním kroku máme jedinou možnost a to* $w = abb$*. V druhém kroku máme* $|w| + 1$ *možností což jsou čtyři možnosti. V posledním kroku to bude* 6 *možností a celkový počet variací řetězců přijímané J jsou* $1 * 4 * 6 = 24$*. Počet všech variací řetězce* $y_1y_2y_3$ *je ale* 6!*.*

Vidíme, že počet všech variací bude obrovský, ale jenom relativně malý počet variací bude přijímáno obecným skákajícím automatem.

Kapitola 4

Určení členství řetězce do jazyka popsaného jednoduchým skákajícím automatem

Zřejmým přístupem pro určení problému členství by bylo prohledávaní stavového prostoru známými metodami jako je prohledávání do hloubky nebo metody zpětného navrácení. Uzel ve stavovém prostoru reprezentuje stav skákajícího automatu a pamatuje si aplikované pravidlo. Navíc nově vzniklý uzel musí obsahovat pozici odstraněného symbolu pro zpětné navrácení. Prohledání celého stavového prostoru bývá často časově náročné a charakteristické pro problémy v třídě složitosti NP. Algoritmus je popsán v kapitole členství řetězce do jazyka popsaného obecným skákajícím automatem 6, kde je tento postup nutný.

Na problém přijímaní řetězce jednoduchého skákajícího automatu můžeme také nahlížet jako hledání způsobu, jak odečíst jednotlivé počty výskytu symbolů v řetězci vzhledem k sekvenci přechodů v automatu. Existuje-li nějaká sekvence přechodů automatu, při kterých jsou odečteny všechny symboly řetězce, pak je řetězec přijímán. Pro sekvence přechodů, která končí v některých z koncových stavů, platí dvě možnosti. Buď jsme provedli nějakou sekvenci přechodů a zůstali jsme ve stejném stavu nebo jsme provedli jeden přechod a pokročili směrem ke koncovému stavu. Tento fakt vede k redukování problému členství řetězce skákajícím automatem na problém o systému lineárních rovnic. V takovém systému pak hledáme alespoň jedno řešení v celých nezáporných číslech.

V této kapitole bude popsán postup převedení jednoduchého skákajícího automatu na systém lineárních diofantických rovnic. Proces převedení problému je popsán v sekci 4.1. Dozvíme se že, pokud JFA obsahuje vnořené cykly, potom problém musíme dále definovat. Rozdělíme problém na JFA s vnořenými cykly a bez vnořených cyklů. Sekce 4.2 popisuje převod libovolného JFA na formuli Presburgerovy aritmetiky. V sekci 4.3 je popsán vztah mezi systémem lineárních rovnic a jazykem přijímaným jednoduchým skákajícím automatem.

4.1 Převod problému na soustavu systémů diofantických lineárních rovnic

Je zřejmé, že řetězec je přijímán jednoduchým skákajícím automatem jenom tehdy, když je celý přečten pomocí sekvencí přechodů a skončí v nějakém koncovém stavu, přičemž nezáleží na pořadí přečtených symbolů. Tím pádem nemusíme nahlížet na řetězec jako

souvislou posloupnost symbolů, ale jako diskrétní hodnoty jednotlivých četností symbolů v řetězci. A při průchodu automatem místo odebíraní symbolů v řetězci, budeme jednotlivé přečtené symboly odečítat z celkového počtu daného symbolu nacházejících v řetězci.

Každý skákající automat můžeme vyjádřit orientovaným multigrafem. Pro jednoduchost představme si tento orientovaný graf jako množinu stavů a přechodů mezi nimi. Pokud na problém přijímaní řetězce budeme přihlížet jako na grafový problém, pak jsme museli začít cestu v počátečním stavu, při přechodu do dalších stavů jsme aplikovali nějaké pravidla a skončili jsme v některém z koncových stavů. Obecně platí, že existuje konečně mnoho cest mezi počátečním stavem a koncovým. Jinými slovy pokud je řetězec přijímán jednoduchým skákajícím automatem, potom existuje konečně mnoho cest, jak při průchodu grafem skončit v koncovém stavu.

Z předchozí kapitoly víme, že existuje minimálně jedna cesta z počátečního stavu do koncového stavu, po které jsme se pohybovali při sekvenci přechodů a tím přečetli celý řetězec. Přemýšlejme nad charakteristikou této cesty. Cesta musí minimálně obsahovat stavy, které tvoří jednoduchou cestu (stavy se neopakují) z počátečního do koncového stavu.

Definice 26 *Nechť je dán JFA* $J = (Q, \Sigma, R, s, F)$*. Přímá cesta je definována jako konečná posloupnost stavů a přechodů mezi nimi jako*

$$p = (p_0, e_1, p_1, ..., e_n, p_n),$$

kde $p_i \in Q$ *a* $e_i \in \Sigma^*$ *pro* $i = 0, 1, ..., n$*. Pro přímou cestu* p *navíc musí platit následující. Prvním členem této posloupnosti je počáteční stav* $p_0 = s$*. Pro poslední člen posloupnosti musí platit* $p_n \in F$*. Pro každou sousední dvojici stavů* p_i, p_{i+1}*, pro* $i = 0, ..., n-1$*, musí existovat alespoň jedno pravidlo pro přechod z prvního do druhého stavu* $p_i y \to p_{i+1} \in R$*. Poslední podmínkou je, že žádný stav v posloupnosti se nesmí opakovat* $p_i \neq p_j$ *s jedinou výjimkou, kdy se první a poslední člen posloupnosti mohou rovnat* $p_0 = p_n$*.*

Je zřejmé, že v nějakém stavu přímé cesty můžeme provést nějakou sekvenci přechodů a poté vrátit se do stejného stavu a dál pokračovat ke koncovému stavu. Cyklus je další prvek, který se může nacházet ve grafu popisující skákající automat. Pro náš problém to znamená, že než se přesuneme dále po přímé cestě, tak provedeme libovolně mnoho sekvencí přechodů cyklů. Pokud se v přímé cestě rovná první a poslední stav, pak je tato přímá cesta zároveň i cyklem. Jinými slovy pokud provedeme sekvenci přechodů v takové přímé cestě, tak první sekvencí přechodů bude přímá cesta a další už budou cykly. Budou nás zajímat elementární cykly, kde se opakuje jenom první a poslední stav. Elementární cyklus je zároveň jednoduchou cestou.

Definice 27 *Nechť je dán JFA* $J = (Q, \Sigma, R, s, F)$*. Elementární cyklus je posloupnost stavů* $c \in Q$ *a přechodů* $e \in \Sigma^*$

$$c = (c_0, e_1, c_1, ..., e_n, c_n),$$

taková, že první stav je shodný s posledním stavem $c_0 = c_n$*, žádné další stavy se neopakují* $p_i \neq p_j$ *a pro každou dvojici sousedních stavů* c_i, c_{i+1} *existuje pravidlo pro přechod do následujícího stavu* $c_i y \to c_{i+1} \in R$*.*

Důležitou vlastností jednoduché cesty je, že jakákoliv cesta je vytvořena z jednoduchých cest.

Lemma 28 *Nechť je definován JFA $J = (Q, \Sigma, R, s, F)$. Nechť P značí množinu všech přímých cest v J a C množinu všech elementárních cyklů v J. Pokud je řetězec $w \in Sigma^*$ přijímán J, pak existuje posloupnost stavů, která lze vyjádřit jako kombinace jedné přímé cesty $p \in P$ a množiny elementárních cyklů C.*

Příklad 29 *Nechť je zadán skákající automat*

$$M = (\{s, q, f\}, \{a, b, c\}, R, s, \{f\}),$$

kde R obsahuje následující pravidla $sb \to q$, $qa \to q$, $qc \to f$, a $fa \to s$ a řetězec $w =$ "$baaca$".

Každá přímá cesta začíná v počátečním stavu v našem případě ve stavu s. Ze stavu s se můžeme dostat přečtením $'b'$ jenom do stavu q, který nepatří do množiny koncových stavů. Ze stavu q se můžeme dostat znovu do stavu q přečtením $'a'$ a do stavu f přečtením $'c'$. Protože se v přímé cestě nesmí opakovat stavy (kromě počátečního), první přímou cestou je tedy posloupnost $p_1 = (s,'b', q,'c', f)$. Druhou přímou cestou je posloupnost $p_2 = (s,'b', q,'c', f,'a', s)$ a zároveň tato cesta je elementárním cyklem. Ve skákajícím automatu M jsou dva elementární cykly $c_1 = (q,'a', q)$, $c_2 = p_2 = (s,'b', q,'c', f,'a', s)$.

Řetězec w je přijímán M, protože jej lze přečíst jako sekvenci přechodů p_1 a $3c_1$.

Další otázkou je, jak pro daný řetězec zjistit, jakou kombinací přímé cesty a množiny cyklů lze řetězec přečíst celý nebo určit, že řetězec nelze žádným způsobem přečíst. Protože nezáleží na pořadí čtení, místo postupného odebíraní symbolů z řetězce, můžeme z celkového počtu výskytu jednotlivých symbolů řetězce odečíst symboly nacházející se v sekvencí přechodů. Jmenovitě pro řetězec w a přímou cestu p odečteme jednotlivé symboly nacházejí se na přímé cestě od počtu výskytu symbolů v w. Pokud dostaneme nulu pro všechny symboly, pak je w přijímán.

Definice 30 *Ohodnocení přímé cesty nebo elementárního cyklu p, označujeme jako $|p|$, je vektor $|p| = (x_1, x_2, ..., x_n)$ pro $n = |\Sigma|$, kde x_i značí počet symbolů $a_i \in \Sigma$ v p. $|p|_a$ značí počet výskytů symbolu a v přímé cestě nebo elementárního cyklu p.*

První podmínkou pro řetězec přijímaný skákajícím automatem je tedy, aby minimálně obsahoval znaky přečtené na cestě od počátečního stavu do libovolného koncového stavu. Jelikož se po přečtení cyklu vracíme na přímou cestou, bude nás zajímat odpověď na otázku jaká kombinace všech cyklů nám po odečtení symbolů dá nulu.

Teorém 31 *Každý úplně deterministický JFA $J = (Q, \Sigma, R, s, F)$. Lze převést na lineární diofantické rovnice.*

Důkaz 32 *Úplně deterministická JFA $J = (Q, \Sigma, R, s, F)$ může obsahovat jenom jednu přímou cestu, která může být i elementárním cyklem. Pokud úplně deterministický JFA obsahuje jenom přímou cestu, potom pro $w \in L(J)$ platí rovnosti $|w|_a = |p|_a$, pro všechny $a \in \Sigma$. Pro elementární cyklus platí rovnosti $|w|_a = x|c|_a$ pro nějaké $x \geq 0$ značící kolikrát byl tento cyklus proveden.*

Příklad 33 *Nechť je definován jednoduchý skákající automat M stejně jako v předchozím příkladu a řetězec $w \in \Sigma^*$. Je zřejmé, že tento automat přijímá řetězec, který vznikl z nějaké kombinace přímých cest a cyklů. Pokud w je přijímán skákajícím automatem M, tak řetězec w musí minimálně obsahovat znaky ohodnocené přímé cesty $|p_1| =$ "bc" nebo $|p_2| =$ "bca".*

Jelikož skákající automat přijímá jakékoli variace řetězce w, tedy nezáleží na pořadí jednotlivých znaků ale na jejich četnosti. Tedy hledáme takovou kombinaci všech cyklů a přímých cest, aby se četnost všech znaků rovnala nula. Obecně tedy nechť neznámá proměnná x_1 značí kolikrát jsme provedli sekvencí přechodů cyklu $|c_1| = a$, a neznámá x_2 značí počet provedených cyklů $|c_2| = bca$. Pokud budeme uvažovat nad první přímou cestou, potom můžeme pro každý znak napsat:

$$x_1|c_2|_a + x_2 * |c_1|_a + |p_1|_a = |w|_a$$
$$x_1|c_2|_b + x_2 * |c_1|_b + |p_1|_b = |w|_b$$
$$x_1|c_2|_c + x_2 * |c_1|_c + |p_1|_c = |w|_c$$

Po dosazení dostaneme:

$$x_1 + x_2 = |w|_a$$
$$x_1 + 1 = |w|_b$$
$$x_1 + 1 = |w|_c$$

Je patrné,že pokud existuje nezáporné celočíselné řešení tohoto systému, pak řetězec w je přijímán skákajícím automatem. Pro úplnost musíme ještě sestrojit další systém lineárních rovnic podle druhé cesty p_2. Druhý systém by se lišil jenom první rovnicí, a to $x_1 + x_2 + 1 = |w|_a$. Tedy pro řetězec platí $w \in L(M)$ jenom tehdy, když existuje alespoň jedno nezáporné celočíselné řešení buď v prvním systému nebo v druhém systému lineárních rovnic.

Definice 34 *Nechť je definován jednoduchý skákající automat $J = (Q, \Sigma, R, s, F)$, potom J lze transformovat na ekvivalentní pětici*

$$J' = (Q, \Sigma, P, C, R'),$$

kde Q a Σ jsou stejně definované jako u JFA, P je množina všech přímých cest $p \in Pp \in Q^n$, kde $n \geq 0$. C je množina všech cyklů $c \in Cc \in Q^n$ a R' je množina pravidel ve tvaru $R' \subset P \times C \cup C \times C$. Každé pravidlo, zapsáno symbolicky jako $p \to q \in$, znamená, že v J existuje pravidlo $xy \to z$, kde stav $x \in p$ a $z \in q$. Píšeme, že z přímé cesty nebo cyklu p je dosažitelné na cyklus p. Nechť je definován uzávěr cyklů γ značící tranzitivní uzávěr jako

$$\gamma(p) = \{c : c \in C, p \to^* c\}, \text{ kde } p \in P.$$

Z každé cesty se nemusíme dostat do všech elementárních cyklů, protože v automatu nemusí existovat pravidla pro přechod. Ke každé přímé cestě převedeme na lineární systém jenom ty elementární cykly, které jsou z této přímé cesty dosažitelné.

Teorém 35 *Nechť je dán JFA jako pětice $J = (Q, \Sigma, P, C, R)$. Každý JFA lze potom převést na soustavu lineárních systémů následovně*

$$S = \{Ax + |p| : \ p \in P, \ A_{ij} = |\gamma(p)_j|_{\Sigma_i}\},$$

pro $i = 0, 1, ..., |\Sigma|$ a $j = 0, 1, ..., |\gamma(p)|$.

Obecně pro každou přímou cestu v JFA platí, že počet lineárních rovnic se bude rovnat počtu všech symbolů ve vstupní abecedě Σ a počet neznámých se bude rovnat počtu elementárních cyklů dosažitelných z přímé cesty. Ve výsledku dostaneme soustavu o q lineárních systémů, kde q značí počet přímých cest. V každém systému lineárních rovnic nás zajímá řešení jenom v nezáporných číslech $X \in \mathbb{N}$, protože počet cyklů nemůže být záporné číslo.

Nalezení řešení v některém z těchto systému nemusí ještě znamenat, že řetězec je přijímán JFA, protože může existovat další závislost mezi dvěma cykly.

4.2 Převod problému na podmínkové celočíselné programování

Pokud v jednoduchém skákajícím automatu existuje vnořený cyklus, potom nelze přečíst vnořený cyklus, aniž bychom nepřečetli alespoň jednou cyklus, který je o úroveň níže oproti vnořenému. Musí platit, že počet provedených přechodů vzhledem k cestě v nezanořeném cyklu musí být minimálně jedna. Nicméně v systému lineárních rovnic může zároveň existovat takové řešení, které nevyhovuje této podmínce. Proto bude nutné buď najít jiné řešení nebo dále definovat tento problém. Naším cílem je nalezení boolovského výrazu, který by korespondoval s závislostmi mezi elementárními cykly.

Definice 36 *Nechť je dán JFA $J = (Q, \Sigma, P, C, R')$. Elementární cyklus $c \in C$ je vnořený elementární cyklus, pokud $c \to q \in R$ a $q \in C$. Pokud existuje v J vnořený cyklus, pak hovoříme o jednoduchém skákajícím automatu s vnořenými cykly, symbolicky značeno jako JFA^{-IC}, jinak J je JFA s vnořenými cykly.*

Lemma 37 *Nechť je dán JFA $J = (Q, \Sigma, P, C, R)$. Nechť $c \in C$ a x_q, x_c jsou korespondující proměnné udávající počet provedených cyklů. Potom pro elementární cyklus c musí být minimálně dodržená podmínka ve tvaru*

$$(\bigvee_{\substack{q \to c \in R \\ q \in C}} x_q \geq 1) \vee x_c = 0$$

Všimněte si, že podmínka je neplatná jenom tehdy, když pro všechny cykly q platí $x_q = 0$ a pro cyklus c platí $x_c \geq 1$. Tímto způsobem můžeme definovat všechny závislosti elementárních cyklů a tím získáme formuli v Presburgerově aritmetice reprezentující JFA. Problém nastává v situaci, kdy jsou dva a více elementárních cyklů na sobě závislých. V tomto případě musíme rozbít jejich závislost tím, že rozšíříme podmínku o závislosti předcházejících elementárních cyklů. Je důležité si uvědomit, že závislosti elementárních cyklů se mění vzhledem k přímé cestě.

Definice 38 *Nechť je dán JFA $J = (Q, \Sigma, P, C, R)$. Potom J lze převést na formuli predikátového logického prvního řadu následovně*

$$\Phi = \Phi_1 \wedge \Phi_2$$

kde Φ_1 jsou lineární podmínky vyjadřující systém lineárních diofantických rovnic, Φ_2 jsou podmínky pro závislost vnořených cyklů pro danou přímou cestu a povolené elementární cykly. Jednotlivé formule jsou formálně definovány jako

$$\Phi_1 = \bigwedge_{a \in \Sigma} \sum_{c \in C} |c|_a x_c + y = |w|_a,$$

$$\Phi_2 = \bigvee_{p \in P} (y = |p| \quad \bigwedge_{c \in C - \gamma(p)} x_c = 0 \quad \bigwedge_{c \in \gamma(p)} \Phi_c),$$

kde y je vektor proměnných $y \in \mathbb{N}^{|\Sigma|}$ vyjadřující povolený počet symbolů přímé cesty a Φ_c je pro přímou cestu p boolovský výraz závislosti elementárního cyklu c na elementární cykly, ze kterých je možné se dostat do cyklu c.

Jinými slovy formule, vyjadřující všechny možné povolené počty symbolů, lze charakterizovat následovně. Pro nějakou kombinaci přímé cesty a elementárních cyklů musí platit ILP Φ_1 definující povolený počet jednotlivých symbolů a zároveň pro přímou cestu nejsou porušeny žádné závislosti elementárních cyklů a nedosažitelné elementární cykly jsou nulové.

4.3 Definování přijímaného jazyka

Definování přijímaného jazyka je nyní jednoduché, když máme k dispozici soustavu systémů lineárních diofantických rovnic popisující JFA. Je zřejmé, že pokud existuje nezáporné celočíselné řešení v některých z systémů v této soustavě, pak je řetězec přijímán. Počet každého symbolu v řetězci musí korespondovat s konkrétní rovnicí v systému a zároveň jsou dodrženy další podmínky.

Teorém 39 *Nechť je dán JFA^{-IC} $J = (Q, \Sigma, R, s, F)$, jeho ekvivalentní soustava lineárních systémů S a řetězec $w \in \Sigma^*$. Nechť $|w| \in \mathbb{N}^{|\Sigma|}$ značí vektor vzniklý z řetězce w, kde každý komponent vektoru x_a koresponduje s počtem symbolů v $|w|_a$ pro $a \in \Sigma$. Pokud existuje vektor $s \in \mathbb{N}^n$ takový, že pro nějaký systém platí $As + |p| = |w| \in S$, pro přímou cestu p, potom $w \in L(J)$. Potom jazyk přijímaný J je definován jako*

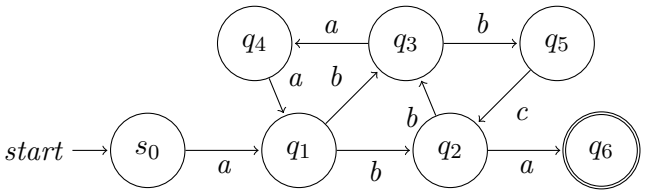
$$L(J) = \{perm(w) : \ |w| = Ax + |p|,\ p \in P,\ |w| \in \mathbb{N}^{|\Sigma|}\}$$

Teorém 40 *Nechť je definován JFA $J = (Q, \Sigma, P, C, R)$ a formule Φ. Pokud ke každé proměnné $x \in N$ v formuli Φ dokážeme přiřadit konkretní hodnotu a zároveň není porušena žádná podmínka v Φ, potom $w \in L(J)$. Potom jazyk přijímaný J je definován jako*

$$L(J) = \{perm(w) : \ |w| \in \Phi,\ |w| \in \mathbb{N}^{|\Sigma|}\}$$

Demonstrujeme dosud získané informace na složitějším příkladu.

Příklad 41 *Nechť je dán jednoduchý skákající automat následovně.*



Potom množina všech cyklů je $C = \{(q_1, q_3, q_4), (q_1, q_2, q_3, q_4), (q_2, q_3, q_5)\}$ a množina přímých cest je $P = \{(s_0, q_1, q_2, q_6), (s_0, q_1, q_3, q_5, q_2, q_6)\}$. Soustava systémů lineárních diofantických rovnic $Ax + B = q$ pro přímé cesty $p1$, $p2$ a řetězec w je definován následovně:

$$S = \begin{matrix} a: \\ b: \\ c: \end{matrix} \begin{bmatrix} 2 & 2 & 0 \\ 0 & 2 & 2 \\ 1 & 0 & 1 \end{bmatrix} + \{ \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} \} = \begin{bmatrix} |w|_a \\ |w|_b \\ |w|_c \end{bmatrix}$$

Protože všechny elementární cykly jsou dosažitelné z přímé cesty, potom neexistuje žádná závislost cyklů. Jazyk L přijímaný skákajícím automatem J pro $p \in P$ a $x \in \mathbb{N}^3$ je definován jako

$$L(J) = \{perm(w) : \ |w|_a \ = 2x_1 + 2x_2 + |p|_a, |w|_b = 2x_2 + 2x_3 + |p|_b, |w|_c = x_1 + x_3 + |p|_c\}$$

Do vektoru x dosadíme do proměnných libovolné nezáporné čísla např. $x = (1, 1, 1)$. Potom přijímaný řetězec může být $w = aaaaaabbbbbcc$.

Doposud jsme rozdělili otázku členství na tři problémy jmenovitě členství v úplně determinovaných JFA, JFA bez vnořených cyklů a JFA. Hledání řešení pro úplně deterministický JFA je jednoduché, protože máme rovnice o jedné neznámé. Hledání řešení pro JFA bez vnořených cyklů pro jednu přímou cestu je ekvivalentní s hledáním řešením lineárního diofantického systému v nezáporných celých číslech. Problém členství pro libovolný JFA je definován pomocí podmínkového lineárního celočíselného programování. Je zřejmé, že hledání řešení v JFA bez vnořených cyklů bude časově méně náročné než hledání řešení pro libovolný JFA.

Kapitola 5

Diskrétní optimalizační problém

Mezi diskrétní optimalizační problémy patří úlohy, kde musíme najít optimální řešení z diskrétní množiny všech možných řešení. Množina všech řešení je většinou spočitatelná. Charakteristickou úlohou pro diskrétní optimalizační problém je hledání podmnožiny objektů z množiny n objektů, kde každý objekt obsahuje svoji celočíselnou hodnotu. Hledaná podmnožina může být popsána tak, že celkový součet všech objektů je jeden milion.

Nejjednodušším postupem bude postupně vytvářet všechny možné kombinace objektů a sčítat jejich hodnoty dokud se nějaká kombinace nebude rovnat hledanému součtu. Ale takové řešení je velmi nepraktické, protože existuje 2^n podmnožin. Pro diskrétní optimalizační problém je charakteristické, že vybíráme nějaké řešení z velkého počtu možností. Takový postup se nazývá metoda hrubé síly. Diskrétní optimalizační problém je často charakterizován diofantickými lineárními rovnicemi a může být dále popsán predikátovou logikou prvního řádu. Proměnné v tomto systému mohou nabývat jenom celočíselné hodnoty.

Je známo, že výpočet řešení pro jedinou diofantickou lineární rovnici, je založenou na Bézoutových koeficientech [20]. Hledání řešení diofantické lineární rovnice v nezáporných číslech je založeno na hledání tzv. Frobeniova čísla. Frobeniovo číslo je největší číslo b, pro kterou rovnice $a_1x_1 + a_2x_2 + ...a_nx_n = b$ nemá řešení.

Systém lineárních diofantických rovnic, zkráceně napsáno jako DLES, je složitější na řešení, nicméně nalezení alespoň jednoho řešení takového systému je důležité v mnoha odvětí v informatice. Neznámější problémy, které lze charakterizovat jako diskrétní optimalizační problém, jsou AC-unifikace, celočíselné lineární programování, podmínkové logické programování, optimalizace kompilátorů, problém obchodního cestujícího, problém batohu atd. V mnoha případech stačí najít jedno řešení nebo kontraindikaci řešitelnosti systému. Většina diskrétních optimalizačních problému patří do skupiny časové složitosti NP úplné.

V jednotlivých sekcí se zabýváme postupem řešení JFA bez vnořených cyklů nebo libovolného JFA definovaného v Presburgové aritmetice. Protože v JFA bez vnořených cyklů hledáme alespoň jedno řešení v soustavě lineárních systému, potom musíme vypočítat všechny DLES. V sekci 5.1 je popsáno řešení DLES založeno na základě transformace matice na trojúhelníkový tvar. Na základě této transformace popíšeme podmnožinu jednoduchých skákajících automatů, které lze vyřešit v polynomiálním čase. Sekce 5.2 vnímá DLES jako optimalizační úlohu v celočíselném lineárním programování. Ukážeme, že pro pevně daný počet lineárních rovnic, lze tento problém vyřešit v polynomiálním čase. Sekce 5.3 řeší formuli v Presburgerově aritmetice a popisuje existující metody. Zajímavou metodou je převedení této formule na konečný automat. Poslední speciální metoda je popsána v sekci 5.4. Problém DLES převede na problém o polynomech.

5.1 Hermitovský maticový tvar

Nejobvyklejší způsob pro výpočet lineárního systému ve tvaru $Ax = B$ v oboru reálných číslech je založen na základě Gaussovy eliminační metody. V konečném počtu kroků elementárních řádkových úprav provedených na matici AB získáme horní trojúhelníkový tvar matice a potom řešení lze jednoduše přečíst z nově vypočítané matice. S analogií Gaussovy eliminační metody můžeme v konečném počtů kroků převést matici AB na horní trojúhelníkový tvar, kde jednotlivé koeficienty matice AB budou celočíselné. Potom taková matice se bude nazývat matice v Hermitovském normálním tvaru. Existuje mnoho algoritmů pro převod matice do Hermitovského normálního tvaru [15] [19].

Definice 42 *Matice A je v Hermitovském normálním tvaru pokud platí*

(i) *A je v horním trojúhelníkovém tvaru*

(ii) *$A_{ij} \in Z$, pro $i = 1, 2, ...m$ a $j = 1, 2, ..., n$*

Lemma 43 *Nechť je definována matice $A \in Z$ a její převod na Hermitovský normální tvar napsáno jako $HNF(A)$. Pro každou matici A s celočíselnými koeficienty existuje unikátní matice $H = HNF(A)$ taková, že $H = UA$, kde U je unimodulární matice.*

Lemma 44 *Počet kroků pro výpočet Hermitovského normálního tvaru je omezen horní hranicí polynomem v dimenzi vstupní matice a prostorová složitost je omezena v binárním velikostí čísel ve vstupní matici [19].*

Z Hermitovského tvaru matice pak můžeme jednoduše zpětnou substitucí proměnných přečíst celočíselné řešení, protože matice je v horním trojúhelníkovém tvaru. Výsledek bude mít několik podob v závislosti na počtu proměnných a zadaných řádku v původní matici. Jedná se o determinovaný, poddeterminovaný a naddeterminovaný systém lineárních rovnic. U determinovaného systému existuje buď žádné řešení nebo právě jedno. U poddeterminovaných systému je nutné při zpětné substitucí zavést nové proměnné a pomocí těchto proměnných vyjádřit vztah vůči původním proměnných. Tímto postupem získáme obecné řešení v oboru celých čísel.

Zajímavou vlastností je, že pokud matice A je čtvercová a zároveň všechny řádky matice jsou lineárně na sebe nezávislé, potom řešením musí být právě jeden bod. Pokud tento bod bude mít celočíselné souřadnice, pak máme řešení i pro DLES.

Lemma 45 *Nehomogenní systém lineárních rovnic $Ax = B$ má právě jedno řešení $x \in N^n$ nebo žádné řešení pokud matice A je čtvercová $n \times n$ a zároveň $det(A) \neq 0$. Pokud $det(A) = 0$, potom systém nemá řešení nebo konečně mnoho řešení.*

Protože matici A dokážeme transformovat v polynomiálním čase, potom existuje podmnožina jednoduchých skákajících automatů, u kterých lze zjistit problém členství v polynomiálním čase pro libovolný řetězec.

Teorém 46 *Nechť je definován JFA $J = (Q, \Sigma, R, s, F)$, kde $P = \{p\}$, $w \in \Sigma^*$ a jeho korespondující DLES $Ax + |p| = |w|$, kde $|w| \in \mathbb{N}^{|\Sigma|}$. Pokud $det(A) \neq 0$, potom můžeme vyřešit problém členství řetězce w v polynomiálním čase.*

Následuje popis známého postupu pro řešení DLES, při kterém získáme jeden konkrétní vektor řešení a bodovou mřížku. Obecné řešení je pak definované součtem vektoru řešení a bodové mřížky.

Teorém 47 *Známou metodou [16] pro výpočet DLES $Ax = B$ v celočíselném oboru je vypočítaní Hermitovského tvaru matice G, která je definována následovně*

$$G = \left[\begin{array}{c|c} A^t & 0 \\ \hline B^t & 1 \end{array}\right].$$

Pokud v systému $Ax = B$ existuje alespoň jedno řešení, potom musí platit

$$HNF(G) = \left[\begin{array}{c|c} C & 0 \\ \hline 0 & 1 \\ \hline 0 & 0 \end{array}\right],$$

kde C se skládá z nenulových řádků. Pokud Hermitovská matice je v tomto tvaru, pak musí platit pro unimodulární matici takovou, že $PG = HNF(G)$

$$P = \left[\begin{array}{c|c} Q_1 & 0 \\ \hline -Y^t & 1 \\ \hline R & 0 \end{array}\right].$$

R je bodová mříž. $-Y$ je vektor řešení DLES $Ax = B$. Obecné řešení v oboru celých číslech je pak definováno jako $-Y + R$.

Aplikováním stejných řádkových operací vzhledem k transformaci na Hermitovskou matici na jednotkovou matici dostaneme unimodulární matici P a jeden konkrétní výsledek.

Všimněte si, že pokud systém má řešení v celých číslech, tak to nemusí znamenat, že existuje řešení i v nezáporných číslech. Vzhledem k povaze našeho problému, hledáme jenom nezáporný celočíselný výsledek, nemůžeme tímto způsobem všechny případy. Nicméně pro speciální případy může být tato metoda optimální. Všechny jednotlivé případy budou vysvětleny následovně ve vztahu s jednoduchým skákajícím automatem.

Výpočtem Hermitovskou matici nelze modelovat to, že některá proměnná se nemůže rovnat nule v závislosti na jiné proměnné.

Teorém 48 *Nechť je definován JFA jako $J = (Q, \Sigma, P, C, R)$ a jeho korespondující DLES $Ax + |p| = |w|$ pro řetězec $w \in \Sigma^*$ a $|w| \in \mathbb{N}^{|\Sigma|}$. V polynomiálním čase lze vyřešit tyto případy*

(i) neexistuje řešení v celých číslech

(ii) neexistuje řešení v záporných celých číslech, ale existuje jenom v kladných

(iii) platí $det(A) \neq 0$ a A je čtvercová matice $n \times n$

(iv) v dalších speciálních případech

Speciálním případem rozumíme, když během transformace matice na Hermitovský tvar získáme unimodulární matici P, která bude obsahovat nezáporný vektor řešení Y. Dopředu nedokážeme určit v jakých případech vznikne nezáporný vektor po transformaci matice na Hermitovský tvar.

5.2 Celočíselné lineární programování

Celá řada praktických problémů týkajících se diskrétní optimalizace může být modelována a řešena pomocí celočíselného lineárního programování (ILP) [20]. Oproti lineárnímu programování jsou v ILP všechny proměnné omezeny pouze na celá čísla. Obecný ILP problém je definován jako matematický model s lineárními vztahy a optimalizační funkcí sloužící k hledání buď největší hodnoty optimalizační funkce nebo nejmenší hodnoty optimalizační funkce v množině definované soustavou lineárních rovností a nerovností.

Definice 49 *Obecný celočíselný lineární problém je definován následovně.*

$$\begin{array}{ll} \textit{minimalizuj} & c_n^T x \\ \textit{vzhledem k} & Ax = b \\ & x \in N_0^n \end{array}$$

Naivním přístupem k vyřešení ILP je, že odstraníme omezení proměnných na celá čísla a vyřešíme korespondující lineární problém (LP). Takovému postupu se říká LP relaxace ILP problému. Klasickým algoritmem pro řešení lineárního problému je polynomiální metoda simplex. Pokud po vyřešení LP relaxace budou některé proměnné reálné čísla, tak je zaokrouhlíme k nejbližšímu celému číslu. Zaokrouhlení může ale způsobit porušení nějaké podmínky. Další nutné úpravy budou muset být provedeny.

Protože hledáme jakékoliv kladné celočíselné řešení, potom optimalizační funkce bude pro naše potřeby nulová. Hlavní nevýhodou ILP je časová složitost. Zatímco úloha lineárního programu metodou simplex je řešitelná v polynomiálním čase, obecná úloha ILP je tzv. NP-těžká.

5.2.1 Metody založené na procházení mnohostěnu

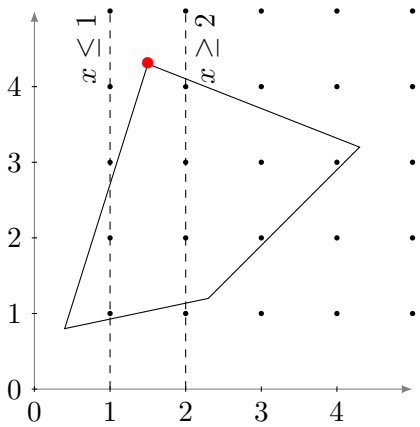
ILP problém ve tvaru $Ax = B, x \geq 0$ tvoří mnohostěn, který je definován jako průsečík konečné mnoho polorovin pro každou lineární nerovnost. Hlavní myšlenkou je prohledávání tzv. přípustné oblasti mnohostěnu, která zahrnuje všechna přípustná celočíselná řešení. Přípustná oblast je konvexní mnohostěn, obsahující všechny body se celočíselnými souřadnicemi. Protože v přípustné oblasti hledáme jenom kladná celá čísla, počet těchto řešení je vždy konečně mnoho. Zajímá nás početní funkce $f(s)$, která spočítá všechny celočíselné body konvexního mnohostěnu. Zajímá nás kolik existuje možností, jak lze přečíst jednotlivé cykly tak, aby jejich počet dal počet všech symbolů v řetězci. Pokud existuje nula způsobu, pak řetězec není přijímán JFA. Tudíž pokud tento mnohostěn obsahuje nula celočíselných bodů, pak $Ax = B$ nemá řešení.

Prvním, kdo se zabýval funkcí pro výpočet počtu bodů v mnohostěnu byl Eugéne Ehrhart [5]. Barvinok [13] nalezl algoritmus pro vypočítání celočíselných bodů uvnitř mnohostěnu. Mnohostěn lze přepsat jako sumu krátkých racionálních funkcí ze kterých, můžeme vyřešit optimilizační otázky jako přípustnost systému, výčet všech řešení nebo i optimalizační otázku. Tento princip dekompozice mnohostěnu na jednoduší tělesa je používán i v novějších algoritmech jako [24]. Pro pevně danou dimenzi, algoritmus může vypočítat počet bodů v mnohostěnu v polynomiálním čase [9].

Tento postup je spíše vhodný pro menší problémy s menším počtem dimenzí, protože počet řešení v přípustné oblasti může být obrovský.

5.2.2 Metoda větvení a mezí

Hlavním principem metody je postupné dělení přípustného prostoru na dva menší. Dělení prostoru je provedeno na základě optimálního vypočítaného bodu LP relaxace problému. Když pro LP relaxaci původního problému není nalezeno optimální celočíselné řešení , pak se původní problém rozdělí přidáním nových mezí. Rozdělení prostoru je provedeno podle proměnné, která porušila celočíselné omezení viz obrázek 5.1.



Obrázek 5.1: Větvení celočíselného linearního problému

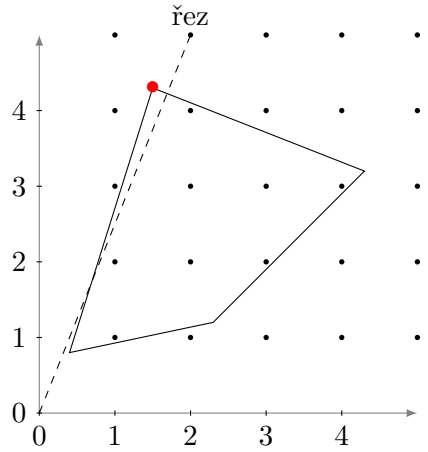
Přípustný prostor na obrázku je definován systémem $Ax \leq B$. Naší optimalizační úlohou je nalézt takový bod, kde y souřadnice je co největší. Červený bod dostaneme vyřešením LP relaxace problému. Protože x i y souřadnice tohoto bodu není celočíselná, musíme problém rozdělit buď podle x nebo y proměnné. Prostor na obrázku je rozdělen podle x na levý prostor nerovností $x \leq 1$ a pravý prostor nerovností $x \geq 2$. Algoritmus postupně větví přípustné prostory na menší, dokud není nalezeno optimální celočíselné řešení nebo žádné neexistuje. Z obrázku je zřejmé, že hledaný optimální bod $[1, 2]$ se nachází v pravé přípustné oblasti.

Během tohoto procesu řešení LP relaxace slouží jako horní hranice a nějaké celočíselné řešení slouží jako dolní hranice. Nově vzniklé systémy jsou reprezentovány jako uzel v binárním stromě. Algoritmus prozkoumává větve binárního stromu, které prezentují podmnožinu řešení. Metoda si uchovává informace o momentálních hranicích a před samotným větvení je použije pro odstranění větve, pokud neobsahuje lepší řešení. Protože naše optimalizační funkce je nulová, hledáme jakékoliv celočíselné řešení, potom tuto část algoritmu můžeme ignorovat.

Pro zvolení proměnné podle které budeme větvit existují různé heuristiky. Nejčastější strategií je, že si zvolíme tu proměnnou u které její desetinný rozvoj je nejblíže k 0.5. Další heuristikou je uchovávání historii zvolených proměnných. Heuristika zvolí tu proměnnou, která bude mít největší dopad na základě uložené historie. Neposlední heuristikou je testování všech kandidátů proměnných na základě toho, která proměnná bude mít nejlepší zlepšení. Tato metoda může být výpočetně náročná. Metoda větvení a mezí patří mezi algoritmy s NP časovou složitostí.

5.2.3 Metoda sečných nadrovin

Metoda sečných nadrovin je založena podobně jako metoda větví a mezí na opakovaném řešení LP relaxace. Získané řešení je testováno, jestli je celočíselné, pokud není, potom existuje lineární nerovnost, která rozděluje optimální bod z přípustného prostoru na přípustný celočíselný prostor. Takové lineární nerovností se říká řez viz obrázek 5.2.



Obrázek 5.2: Řez oddělující přípustnou oblast od skutečné celočíselné přípustné oblasti

Gomoryho řezy jsou efektivně generovány z tabulky algoritmu simplex, kdežto mnoho jiných řezů je generováno až v exponenciálním čase. Řez má dvě důležité vlastnosti

(i) Neobsahuje optimální přípustné řešení pro relaxaci LP problému.

(ii) Obsahuje všechny celočíselné přípustné řešení původního ILP problému. Řez neodstranil žádné celočíselné řešení.

Metoda sečných nadrovin je nepraktická a neefektivní, protože mnoho řezů je nutné aplikovat pro nalezení řešení. Metoda konverguje k celočíselnému řešení a tudíž je nutné řešení postupem času zaokrouhlovat. Věci se změnili, když se zjistilo, že metoda s kombinací metody větvení jsou velmi efektivní.

5.2.4 Metoda větvení a sečných nadrovin

Metoda větvení a sečných nadrovin modifikuje klasickou metodu větvení a to tak, že před větvením přípustného prostoru je provedeno zúžení přípustného prostoru metodou sečných nadrovin. Řezy mohou mít globální charakter, tedy jsou platné pro všechny celočíselné řešení nebo lokální, které platí pro podmínky z právě počítaného podstromu metody větvení a mezí. Metoda sečných nadrovin výrazně redukuje velikost binárního stromu metody větvení. Většina komerčních nástrojů pro výpočet ILP problému používá kombinaci předchozích metod.

5.3 Celočíselné lineární programování rozšířené o logické podmínky

Formule definované v predikátové logice prvního řádu pro strukturu $\langle N, 0, \leq, + \rangle$ jsou známe jako Presburgova aritmetika. Presburgerova aritmetika je rozhodnutelná aritmetika [21], ale v nejhorším případě v dvojité exponenciální časové složitosti [7]. Naštěstí pro naše účely Presburgerova aritmetika bez kvantifikátorů je řešena v exponenciálním čase. Jediná povolená lineární operace je sčítaní. Násobení proměnné koeficientem můžeme jednoduše převést na sčítaní proměnné, protože například $3x = x + x + x$. Všimněte si, že pokud předchozí struktura obsahuje navíc operaci $*$ mezi proměnnýma, potom tato struktura není rozhodnutelná [3]. Pro Presburgerovu aritmetiku platí, že je rozhodnutelná, nesporná a úplná.

Cílem je najít algoritmus, který rozhodne, jestli daná formule v Presburgove aritmetice je přípustná. Takový algoritmus se nazýva problém členství, který přesně rozhodne jestli formule je pravdivá nebo nepravdivá. Existují tři možné způsoby řešení problému přípustnosti Presburgovi aritmetiky bez kvantifikátorů jmenovitě Cooperova metoda [4], metody založené na rozklad do podproblému ILP a na převodu formule na konečné automaty.

5.3.1 Výpočet založen na konečných automatech

Přirozená čísla mohou být vnímána jako řetězce abecedy $\{0, 1\}$. Budeme je psát zleva doprava tzn., že jejich nejvýznamnější bit je posledním symbolem v řetězci. Protože hledáme jenom nezáporná čísla, nemusíme binární reprezentaci kódovat do doplňkového kódu. Binární reprezentace není unikátní, protože můžeme přidat konečně mnoho nul napravo.

Množina celočíselných vektorů, definované Presburgovou aritmetikou, jsou rozpoznatelné konečným automatem respektive konečným automatem s n čtecími hlavami pro vektor $v \in N^n$. Myšlenkou převodu Presburgovi aritmetiky na konečné automaty, reprezentující množinu řešeních, se poprvé zabýval Büchi. Od té doby bylo provedeno dalších výzkumů [2] [26]. Výhodou převedení lineární celočíselných podmínek na konečné automaty je, že můžeme využít známé algoritmy z teorie konečných automatů.

Pomocí reprezentace čísel v binárním kódu popíšeme jeden z možných způsobu převodu formule Presburgerovy aritmetiky na konečné automaty.

Definice 50 *Konečný automat s n čtecími hlavami je pětice*

$$C_n = (Q, \Sigma, R, s, F),$$

kde Q, Σ, s, F je definováno stejně jako u konečných automatů. R je konečná množina relací $Q \times \Sigma^n \times Q \in R$, které symbolický píšeme jako $q(a_1, a_2, ..., a_n) \to p$, kde $p, q \in Q$ a $a_1, a_2, ..., a_n \in \Sigma^*$ a $|a_1| = |a_2| ... = |a_n|$. Nechť jsou definovány řetězce $w_1, w_2, ..., w_n \in \Sigma^*$, píšeme jako $w^n = (w_1, w_2, ..., w_n)$, potom jeden výpočetní krok je definován jako $qaw \Rightarrow pw \in R$, kde $q, p \in Q$, $a \in \Sigma^n$, pro pravidlo $pa \to q$.

Přijímaný jazyk C_n je definován jako

$$L(C_n) = \{w^n : \ w_i \in \{0, 1\}^*, \ sw^n \Rightarrow^* f, \ f \in F\}$$

Takový automat je možné rozdělit na n konečných automatů a spojit je synchronizační funkcí. Průchodem jedním z těchto automatů získáme ostatní čísla ve vektoru, takové že budou vyhovovat DLES. Principem je převést formuli na automat s n čtecími hlavami

a poté zjistit jestli jazyk přijímaný tímto automatem není prázdná množina respektive koncový stav je dosažitelný.

Nyní ukážeme převod problému na konečný automat.

Definice 51 *Nechť je definován konečný automat s n čtecími hlavami* $C_n = (Q, \{0,1\}, R, s, F)$ *a diofantická lineární rovnici ve tvaru* $a_1x_1 + a_2x_2 + ...a_nx_n = b$. *Do množiny stavů přidáme nový stav* $Q = \{q_b\}$ *a tento stav bude počátečním stavem* $s = \{q_b\}$. *Pro každý nový stav* $q_c \in Q$ *najdeme všechny vektory* θ^n *takové, že je splněna rovnost* $a_1\theta_1 + a_2\theta_2 + ... + a_n\theta_n = c \bmod 2$ *a* $\theta_i = \{0,1\}$ *pro všechny* $i = 1, 2, ..., n$. *Pro všechny vektory* θ^n *splňující předchozí podmínky, vypočítáme* $d = \frac{c - (\theta_1a_1 + \theta_2a_2 + ... + \theta_na_n)}{2}$ *a vložíme nový stav* $Q = Q \cup q_d$ *a nové pravidlo* $R = R \cup q_c\theta^n \to q_d$. *Množina koncových stavů bude* $F = \{q_0\}$.

Definice 52 *Nechť je definovaná nerovnost* $ax \leq b$. *Převod na konečný automat* $C_n = (Q, \Sigma, R, s, F)$ *je podobný jako u rovnosti s dvěma rozdíly. Pro každý stav* $q_c \in Q$ *a pro všechny vektory* θ^n *přidáme nový stav* q_d *pro* $d = \lfloor \frac{c - (\theta_1a_1 + \theta_2a_2 + ... + \theta_na_n)}{2} \rfloor$ *a pravidlo* $q_c\theta^n q_d$. *Množina koncových stavů* $F = \{q_c : \ c \leq 0\}$.

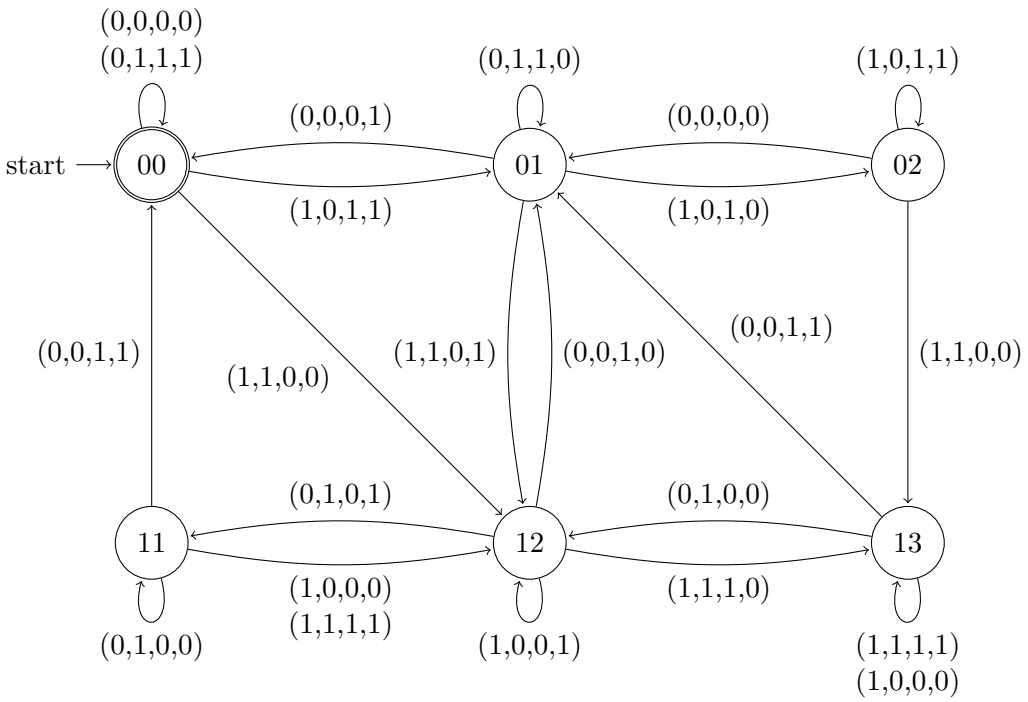
Existují i další způsoby reprezentace diofantických rovnic konečnými automaty. Tento způsob převodu je přímý a jednoduchý [2]. Logické spojky ve formuli Presburgerovy aritmetiky potom můžeme snadno převést následovně.

Definice 53 *Nechť je dána formule* $a_1x_1 + a_2x_2 + ... + a_nx_n = b \wedge c_1x_1 + c_2x_2 + ... + c_nx_n = d$. *Jednotlivé lineární rovnice převedeme algoritmem na konečné automaty* F_1 *a* F_2. *Potom převedení formule na konečný automat* F *je definováno jako* $F = F_1 \cap F_2$.

Definice 54 *Nechť je dána formule* $ax = b \vee cx = d$. *Lineární rovnice jednotlivě převedeme algoritmem na konečné automaty* F_1 *a* F_2. *Potom převedení formule na konečný automat* F *je definováno jako* $F = F_1 \cup F_2$.

Teorém 55 *Nechť je definována formule Presburgerovy aritmetiky bez kvantifikátorů* Φ. *Nechť je definován konečný automat s n čtecími hlavami* N, *který vznikl převodem jednotlivých lineárních diofantických rovnic na konečné automaty a jejich průniku nebo spojení. Pokud jazyk* $L(N) = \emptyset$, *potom formule* Φ *nemá žádné řešení.*

Při tomto procesu nemusíme pokaždé konstruovat konečný automat pro různou hodnotu pravé strany rovnic. Pro počet jednotlivých symbolů můžeme zavést další proměnné a převést tuto formuli na konečné automaty. V takovém automatu se pak můžeme ptát na to, jestli pro zadaný počet jednotlivých symbolů, existuje přijímaný vektor. Na obrázku 5.3 je zobrazený konečný automat s 4 čtecími hlavami pro formuli Presburgerovy aritmetiky $\Phi = x + y = z \wedge 3x + y = w$. Proměnná z nám může například vyjadřovat počet a symbolů v řetězci a proměnná z zase počet b. Pokud za ně dosadíme například $z = 20$ a $w = 40$, potom v korespondujícím automatu se ptáme na to, jestli 3. čtecí hlava přijímá řetězec 001010 a 4. čtecí hlava zároveň přijímá řetězec 000101. Vskutku můžeme najít takovou sekvenci pohybů a to jmenovitě $00, 00, 12, 01, 12, 01, 00$. Při tomto procesu nezáleží na prvních dvou proměnných.



Obrázek 5.3: Řešení formule Presburgerovy aritmetiky $x + y = z \wedge 3x + y = w$

Tento automat přijímá všechny vektory pro tuto formuli. Na průchod tímto automatem může být nahlíženo jako, že z čteného řetězce postupně odstraňujeme jednotlivé symboly.

Procedura převodu Presburgerovy aritmetiky je neefektivní zejména, když formule obsahuje mnoho proměnných [2]. Výpočet je proveden v exponenciálním čase. Nicméně jakmile vypočítáme obecné řešení, můžeme problém redukovat na to, jestli část vektoru je přijímána.

5.3.2 Dekompozice problému na ILP problémy

Většina aplikací pro podmínkový celočíselný lineární problém je založeno na rozložení problému na menší podproblémy pomocí metody větvení a mezí. Další možností je převedení formule do disjunktního normálního tvaru. Každý podproblém vzniklý rozložením je ILP problém. Pokud jeden podproblém má řešení, potom celá formule má řešení.

Před výpočtem ILP podproblému můžeme aplikovat další heuristiky. Každou proměnnou spojíme s doménou přípustných hodnot. Po zvolení dalšího podproblému je pro každou proměnnou redukována doména pomocí jednotlivých podmínek. Pro všechny proměnné nacházející se v podmínce je provedena redukce domény řešeních a to tak, že jsou vyloučeny všechny hodnoty, které nejsou přípustné vzhledem k podmínce a tudíž nejsou řešením. Například mějme proměnnou z a podmínku $z = 2$. Je zřejmé, že přípustná doména pro proměnnou je $z = [2]$. Pokud bude existovat v problému podmínka např. $z > 3$, pak bude přípustná doména prázdná.

Algoritmus propagace domény [1] potom může vypadat následovně. Algoritmus uchovává propagační frontu proměnných. Když je hodnota proměnné modifikována, daná proměnná je vložena na konec fronty, pokud už v ní není. Dokud fronta je neprázdná, algoritmus odstraní první proměnnou z fronty. Potom může algoritmus redukovat doménu proměnné na základě podmínek vázané k této proměnné. Podmínka může být vázaná i k dalším pro-

měnným, tudíž pokud budou modifikovaný tyto proměnné, potom jsou vloženy do fronty. Algoritmus se zastaví ve dvou situacích. V první situaci všechny domény obsahují přípustné hodnoty vzhledem k podmínkám. V druhém případě některá z domén bude prázdná.

Nevyhovující podproblémy jsou analyzovány abychom, zjistili proč, došlo k nepřípustnosti podproblému a tyto informace jsou použity pro získaní dalších podmínek.

5.4 Polynom

Naší strategií bude převést ILP problém $Ax = B$ na problém o polynomech. Vypočítat tzv. Gröbnerovu bázi [18] pro vyřešení problému s polynomy a poté převést řešení zpátky na lineární celočíselné řešení. Začneme s převedením ILP problému na problém s polynomy.

Pro každý řádek původního ILP problému zavedeme novou proměnnou z. Pro každou lineární rovnici dostaneme polynom ve tvaru $z_1^{a_{i1}x_1+...+a_{in}x_n}$ pro $i = 1, 2, ...m$. Nechť od této chvíle vektor A_i značí i. sloupec matice A a nechť je definován polynom z^{A_i} jako $z^{A_i} = z_1^{A_{1i}} z_2^{A_{2i}} ....z_m^{A_{mi}}$. Systém lineárních rovnic můžeme přepsat jako součin levých a pravých stran těchto polynomů.

$$(z^{A_1})^{x_1}(z^{A_2})^{x_2}...(z^{A_n})^{x_n} = z^B$$

Na tuto rovnici můžeme nahlížet jako na mapovací funkci $k[x_1, x_2, ..., x_m] \mapsto k[z_1, z_2, ..., z_n]$ respektive $y_j \mapsto z^{A_j}$, tudíž ideál generující všechny polynomy řešení je definován následovně.

$$K = \langle x_j - z^{A_j} : j = 1, ..., n \rangle.$$

Celý postup odvození je uveden v [18].

Dalším krokem bude vypočítaní standardní báze G generující množinu ideálu v okruhu polynomiálů $K[x_1, x_2...x_n, z_1, z_2, ..., z_m]$. Standardní báze umožňuje zjistit mnoho důležitých vlastností daného ideálu. Je hlavním aspektem pro vypočítání polynomiálních rovnic. Nejznámějším algoritmem pro převod ideálu na standardní bázi je Buchbergův algoritmus a ostatní metody jako F5 [22], které jsou založeny na stejném principu jako Buchbergův algoritmus, ale jsou efektivnější, protože používají další heuristiky. Všechny operace spojené se standardní bází požadují zvolení pořadí monomů. Většinou existuje více standardních bázi pro nějaké pořadí monomů.

Dalším krokem bude zjistit, jestli polynom $x_1^{b_1} x_2^{b_2}, ..., x_n^{b_n}$ patří do $G[y_1, y_2, ...y_n]$. Zjištění je ekvivalentní s tvrzením, že polynom x^B se dá redukovat na polynom h, pro který platí $h \in k[y]$. Koncept redukce, také může být definované jako Euklidovské dělení polynomů s více proměnnými, je důležitou operací pro výpočet standardní báze.

Definice 56 *Nechť jsou dány dva polynomy f a g, potom f je redukovatelné g, pokud nějaký monom m v f je násobek vedoucího monomu g značeno jako $lm(g)$. Redukce je potom definována jako $f - \frac{m}{lm(g)} * g$.*

Cílem redukce je, aby výsledný polynom neobsahoval monom m. Operaci musíme dál definovat, protože pokud je více monomů v f násobkem, musíme vybrat nějaký monom. Nejlepší je zvolit největší podle pořadí monomů, protože redukce může znovu vytvořit monom, který byl již odstraněn.

Teorém 57 *Nechť je definován $Ax = B$ a jeho korespondující báze polynomů K. Pokud v $Ax = B$ existuje nezáporné celočíselné řešení x_1, x_2, x_m, potom zbytek vzniklý dělením součinu $x_1^{c_1} x_2^{c_2} ...x_n^{c_n}$ se standardní bází G je h. Pokud $h \notin k[x_1, ..., x_m]$, potom $Ax = B$ nemá řešení $x \in \mathbb{N}^m$. Pokud $h = x_1^{q_1}, ...x_n^{q_n}$, potom vektor q je řešením $Ax = B$. [18]*

Příklad 58 *Ideál generující všechny řešení pro DLES* $x_1 + x_2 = 20$ *a* $3x_1 + x_2 = 40$ *je* $K = \langle x_1 - z_1^1 z_2^3, x_2 - z_1^1 z_2^1 \rangle$. *Standardní báze je* $G = \{z_2^2 x_2 - x_1, z_1 x_1 - z_2 x_2^2, z_1 z_2 - x_2\}$. *Snažíme se zjistit, jestli pro zbytek h vznikly redukováním polynomu* $f = z_1^{20} z_1^{40}$ *standardní bází G platí* $h \in k[x_1, x_2]$. *V prvním kroku lze polynom f lze redukovat polynomem* G_3. *První redukce bude tedy* $f = z_1^{20} z_2^{40} - z_1^{19} z_2^{39}(z_1 z_2 - x_2) = z_1^{19} z_2^{39} x_2$. *Nově vzniklý polynom můžeme dále redukovat. Po provedení všech redukcí* G_3 *dostaneme polynom* $z_2^{20} x_2^{20}$, *který dále můžeme postupně redukovat polynomem* G_1. *Konečným výsledkem bude* $h = x_1^{10} x_2^{10}$ *a protože pro h platí* $h \in k[x_1, x_2]$, *potom původní problém má řešení v nezáporných číslech, a to jmenovitě* $x_1 = 10$ *a* $x_2 = 10$.

Převod na standardní bází patří mezi problémy s dvojitou exponenciální časovou složitostí. Všimněte si, že K je odvozen jenom z levé strany rovnic. Tudíž standardní bázi můžeme vypočítat dopředu a jenom jednou.

Kapitola 6

Určení členství řetězce do jazyka popsaného obecným skákajícím automatem

U přijímání řetězce obecného skákajícího automatu dochází k několika problémům, které zvyšují nedeterminovanost. Délka slova může být libovolná, proto může docházet k překrývání řetězců. Toto zvyšuje počet možných pravidel, které můžeme aplikovat pro danou konfiguraci, a zároveň zvyšuje míru zpětného navrácení. Dalším problém je, že vzhledem k povaze skákajícího automatu nemůžeme předem vědět jaké pravidlo máme použit jako první za předpokladu, že z aktuálního stavů existuje více pravidel pro přechod do jiného stavu. Neposledním problémem je, že pokud aplikujeme pravidlo, tak nám z původního řetězce vznikne nový řetězec, ve kterém může vzniknout nové slovo.

U obecného skákajícího automatu můžeme využít stejnou myšlenku jak u jednoduchého skákajícího automatu. Pokud je řetězec přijímán, musí existovat nějaká variace elementárních cyklů a přímých cest. Takovou variaci můžeme klasicky vypočítat ze systému lineárních rovnic respektive z formule v Presburgerově aritmetice. Nicméně ne všechny permutace řetězce jsou přijímány variací elementárních cyklů a přímých cest.

Většina problému v informatice nejsou řešitelné v polynomiálním čase. Když se potýkáme s takovým problémem, zbývá nám jenom několik možností vzdát se, zkusit hrubou sílu nebo se spokojit přibližné řešení. Nezbývá nám nic jiného, než použít metodu zpětného navrácení.

Naším hlavním cílem bude popsání algoritmu pro určení členství řetězce do jazyka popsaného obecným skákajícím automatem. V sekci 6.1 je popsáno rozdělení obecného skákajícího automatu na jednodušší automaty s jednou přímou cestou. V sekci 6.2 představíme algoritmus pro vyhledávání stavového prostor obecného skákajícího automatu. Sekce 6.3 vysvětluje techniku zapamatování. Sekce 6.4 se zabývá heuristikou hladového vyhledávání elementárních cyklů. V sekci 6.5 se pokusíme o charakterizování jazyka přijímaného obecným skákajícím automatem.

6.1 Rozdělení problému

Obecný skákající automat může být obrovský respektive může mít mnoho pravidel a stavů, potom je otázkou, jestli je nutné prohledávat všechny pravidla, nebo jestli můžeme některá pravidla vyloučit předem. Z jednoduchého skákajícího automatu víme, že aby řetězec byl

přijímán JFA, potom existuje minimálně jedna kombinace elementárních cyklů a přímé cesty takové, že dokážeme odstranit všechny symboly z řetězce vzhledem k této kombinaci. Stejnou myšlenku můžeme aplikovat i na obecné skákající automaty.

Teorém 59 *Nechť je definován GJFA jako $J = (Q, \Sigma, P, C, R)$. GJFA J lze převést na soustavu systémů respektive na formuli Presburgerovy aritmetiky Φ.*

Důkaz 60 *Transformace je provedena s analogií převodu JFA na Φ.*

Lineární diofantický systém jasně definuje počet povolených jednotlivých symbolů. Tuto skutečnost můžeme využít a aplikovat na obecné skákající automaty. Jinými slovy pro každý GJFA dopředu dokážeme určit, kolik jednotlivých symbolů může být v přijímaném řetězci. Samozřejmě pro GJFA to ještě neznamená, že řetězec je přijímán, protože jenom některé permutace řetězce s povoleným počtem jednotlivých symbolů vzhledem k DLES jsou přijímány. GJFA můžeme rozložit na několik menších GJFA s jednou přímou cestou a několik elementárních cyklů. Je zřejmé, že pokud existuje řešení v některém korespondujícím DLES menšího GJFA, potom původní GJFA bude mít také řešení.

Teorém 61 *Nechť je definován GJFA jako pětice $J = (Q, \Sigma, C, P, R)$ a řetězec $w \in L(J)$. Pokud rozložíme J na menší GJFA s jednou přímou cestou jako*

$$J = \{J_1, J_2, ..., J_{|P|}\},$$

potom pro řetězec w musí platit $w \in L(J_i)$ pro nějaké $i = 1, 2, ..., |P|$. Navíc pro J_i a jeho korespondující DLES platí $Ax + |p| = |w|$, kde $|w|, |p| \in \mathbb{N}^n$.

Místo prohledávání celého GJFA, nejdříve zjistíme, pro kterou přímou cestu existuje povolená kombinace jednotlivých symbol vzhledem ke vstupnímu řetězci. Problém je rozdělen na několik menších podproblému. Místo prohledávání jednoho velkého automatu můžeme prohledávat několik menších a to sekvenčně nebo paralelně. Jednotlivé podproblémy, reprezentující GJFA, můžeme spojit do jednoho GJFA nebo je řešit postupně.

Všimněte si, že podproblém se skládá jenom z jedné přímé cesty a několika elementárních cyklů. O průchodu v tomto automatu můžeme říct, že buď po přečtení elementárního cyklu zůstaneme ve stejném stavu nebo se posuneme směrem ke koncovému. Za předpokladu, že přímá cesta není zároveň elementárním cyklem, můžeme říct, že jsme se posunuli za hranice elementárního cyklu a k tomuto cyklu se už nikdy nevrátíme.

Tento postup vede k následujícímu poznatku.

Teorém 62 *Nechť je definován GJFA jako pětice $J_1 = (Q, \Sigma, C, P, R)$, řetězec $w \in \Sigma^*$, jeho korespondující systém jako $Ax + p = |w|$, kde $|w| \in \mathbb{N}^n$. Pokud DLES je neřešitelný, potom problém členství řetězce w lze vyřešit jako ILP problém respektive jako problém definovaný ve formuli Presburgerovy aritmetiky.*

Když jsme redukovali hledaný prostor, můžeme přejít k definování algoritmu zpětného navrácení.

6.2 Zpětné navrácení

Při čtení slov v řetězci musíme provést sekvenci rozhodnutí nad množinou pravidel, kde nemáme dostatek informací pro zvolení toho správného pravidla. Každé aplikování pravidla vede k nové podmnožině kandidátů. Některé sekvence zvolených operací, mohou být řešením problému. Metoda zpětného navrácení je metodická cesta testování všech variací sekvencí rozhodnutí, dokud není nalezeno správné řešení. Na rozdíl od hledání řešení tzv. hrubou silou, kde postupně generujeme všechny možné kombinace a ověřujeme jejich pravdivost, metoda zpětného navrácení v každém kroku ověřuje všechny podmínky. Pokud jsou všechny dodrženy, potom metoda pokračuje generováním podmnožiny řešení. Pokud je nějaká podmínka porušena, vracíme se zpátky a pokračujeme jinou cestou. Algoritmus prohledává stavový prostor.

Vzhledem k povaze našeho problému, stav v našem prostoru musí obsahovat informace z jakého stavu jsme přešli do následníka, jaké pravidlo jsme aplikovali a navíc pozici v řetězci, kde jsme odebrali slovo. Uchovávat tuto pozici je nezbytné pro vrácení změn provedených na řetězci.

Odebírání slova může probíhat kdekoliv v řetězci, proto můžeme zvolit strategii pro vybírání slov. Přímá strategie by byla postupně odebírat slova zleva doprava nebo zprava doleva dle jejich výskytů v řetězci. Další strategií je odebírat slovo uprostřed řetězce a poté střídat hledání vlevo a vpravo. Pro tuto strategii si musíme zapamatovat dvě pozice poslední levou a poslední pravou pozici. Od levé pozice hledáme slova nalevo a od pravé na vpravo a postupně hledání střídáme.

Z předchozí sekce jsme si rozdělili problém na několik menších automatů s jednou přímou cestou.

V našem algoritmu si musíme pamatovat pozice postupně odstraněných slov. Budeme si je ukládat do speciálního zásobníku. Hodnota -1 bude znamenat, že hodláme aplikovat nové pravidlo a hledáme slovo od počáteční pozice v řetězci. Pravidlo můžeme odstranit až potom, co je přečten celý řetězec.

Další potřebnou strukturou je zásobník stavů pamatující, v jakém pořadí probíhalo čtení stavů a pravidel. Prohledávání je pak popsáno následovně. Do zásobníku stavů vložíme první stav a jeho první přidružené pravidlo. Ze zásobníku stavů vezmeme poslední stav,pravidlo a pokusíme se ho aplikovat nalezením a přečtením slova. Pokud uspějeme, vložíme do zásobníku stavů následující stav a jeho první pravidlo. Když jsme aplikovali všechny možné pravidla pro daný stav, potom můžeme odstranit stav ze zásobníku. Všechny tyto kroky opakujeme, dokud je zásobník stavů neprázdný nebo dokud jsme nepřečetli celý řetězec a neskončili v koncovém stavu. Všimněte si, že v tomto automatu je jenom jeden koncový stav.

Toto vede k následujícímu algoritmu.

Algorithm 1 Pseudokód pro určení členství řetězce do jazyka popsaného $GJFA$

Input: řetězec w
Output: true pokud w je přijímán $GJFA$, jinak false

```
nechť P je prázdný zásobník pro ukládání pozic přečtených slov
nechť QR je zásobník stavů a pravidel k aplikování
vlož do QR počáteční stav a jeho první pravidlo
while zásobník QR je neprázdný do
    nechť q je poslední stav v QR
    nechť r je poslední pravidlo v QR ve tvaru qy → q'
    nechť p je posledni pozice v P
    if pokud q je koncový stav a w je prázdný řetězec then
        return true
    end if
    if existuje pravidlo r then
        if na poslední pozici v P není -1 then          ▷ musíme přečíst jiné slovo
            vrátíme y na pozici p do w
        end if
        if nechť pro pozici p' platí p' > p a existuje slovo v y v w na pozici p' then
            odstraníme z w slovo y nejblíže k pozici p + 1
            poslední hodnotu v P nastavíme na pozici odstraněného slova
            do QR vložíme stav q' a jeho přidružené pravidlo
            do P vložíme pozici −1
        else
            posuň se o další pravidlo r dopředu
            nastav poslední pozici v P na −1
        end if
    else
        odstraň ze zásobníku QR poslední položku
        odstraň ze zásobníku P poslední pozici
    end if
end while
return false
```

Uvedený algoritmus patří do třídy složitosti NP úplná. Princip zpětného navrácení se používá pro velké množství problému, kde není znám efektivnější algoritmus. Zpětné navrácení lze výrazně urychlit vhodnou heuristikou.

6.3 Ukládání vyřešených podproblémů

V předchozím algoritmu bude docházet k opakovanému prohledávání již prozkoumaného podproblému. Pokud například v libovolném stavu pro řetězec aplikujeme všechny jeho pravidla a zjistíme, že všechny další variace přečtených slov z tohoto stavu nejsou přijímány. Je pravděpodobné, že v nějakém dalším kroku výpočtu se dostaneme do stejného stavu, se stejným řetězcem, tudíž výpočet tohoto podproblému skončí zase neúspěchem. Navíc tento výpočet stejného podproblému může vzniknout vícekrát. Jinými slovy ten stejný výpočet je opakovaně prováděn a tím dramaticky zpomaluje předchozí algoritmus.

Stěžejní myšlenkou je rozklad problému na podproblémy, které jsou řešeny a jejich řešení je ukládáno pro další potenciálně možné použití. Metoda je obzvláště vhodná na úlohy, které se dají dělit na podproblémy, které jsou si podobné a mohou se opakovat.

Příklad 63 *Mějme řetězec $w = aaaaaaaaabab$. Naším úkolem bude přečíst nejdříve slovo a a poté slovo bb. Pokud budeme odstraňovat slova zleva doprava, potom bude desetkrát odstraněno a a zpátky navráceno do řetězce, dokud se neodstraní poslední a a nevznikne slovo bb.*

Nejjednodušším řešením je pro každý stav vytvořit asociativní pole řetězců a před aplikováním pravidla v nějakém stavu vyhledáme právě čtený řetězec v asociativním poli. Pokud v asociativním poli nebude, tak ho tam přidáme a pokračujeme. Jinak tento podproblém nebudeme prohledávat. Nevýhodou je velký nárok na paměťové zdroje, zejména když máme pro každý stav vytvořené asociativní pole. Vzhledem k předchozímu algoritmu můžeme zmenšit počet asociativních polích a to tak, že asociativní pole bude vytvořené jenom pro jednoduché cesty. Tím se o něco zmenší požadavek na paměťové zdroje.

Problém v předchozím příkladu lze řešit i dalším způsobem. Pro každé dvě pravidla se nejdříve pokusíme najít obě dvě slova. Pokud druhé slovo nenalezneme, tak místo navracení slova do řetězce, budeme hledat takové slovo, které vytvoří druhé slovo. Takový postup můžeme použít ve funkci next místo pamatování si všech podproblému a ušetříme paměť za cenu opakovaní některých podproblémů ve funkci next.

6.4 Hladové vyhledávání cyklů

Další strategií je postupně čtení slov takovým způsobem, že se snažíme nejvíce krát přečíst stejný elementární cyklus. Až když nelze daný elementární cyklus přečíst, budeme číst další elementární cyklus nebo postoupíme dál směrem ke koncovému stavu. Tímto způsobem v mnoha případech přečteme daný cyklus vícekrát než je ve skutečnosti potřeba. Vzhledem k ILP problému DLES můžeme předem vypočítat maximální počet jednotlivých cyklů. Pokud přečteme například n krát nějaký cyklus a poté budeme číst další cyklus, můžeme vypočítat vzhledem k n kolikrát maximálně můžeme přečíst následující cyklus a tím korigovat počet přečtených cyklů. Opakovaný výpočet maximálních povolených počtů pro cyklus je neefektivní.

6.5 Jazyk

Jazyk GJFA nelze jednoznačně určit, ale naštěstí stejně jako u JFA, jednotlivý počet výskytu jednotlivých symbolů musí korespondovat s rovnicí v korespondujícím DLES. Tudíž máme nezbytnou podmínku pro jazyk přijímaný GJFA.

Teorém 64 *Nechť je dán obecný skákající automat $M = (Q, \Sigma, R, s, F)$ a jeho korespondující formule Φ.*

$$L(J) = \{\textit{některé } w \in \Sigma^* : \ |w| \in \Phi, \ |w| \in \mathbb{N}^n\}$$

Taková podmínka ale nebude dostačující. Další nezbytnou podmínku můžeme jednoduše vyvodit z pravidel GJFA. Přijímaný řetězec GJFA vznikl postupným vkládáním slov do prázdného řetězce vzhledem k reverzovanému GJFA. Při každém vložení slova vznikne nový řetězec, který musí začínat nebo končit počátečním symbol jakéhokoliv předtím aplikovaného pravidla.

Teorém 65 *Nechť je dán GJFA* $G = (Q, \Sigma, R, s, F)$ *a řetězec* $w \in \Sigma^*$. *Pro* w *platí* $w \notin L(J)$ *pokud* $qz \to p \not\subset R$ *pro* $z_1 = w_1$ *nebo* $z_n = w_m$ *pro* $n = |z|$ *a* $m = |w|$.

Předchozí podmínky jsou nezbytné to znamená, že některé variace řetězce mohou splňovat tyto podmínky, ale zároveň nejsou přijímány GJFA. Otevřenou otázkou je, jestli existuje dostačující podmínka pro GJFA, která jednoznačně definuje jeho jazyk. Jiným pohledem se můžeme ptát na algoritmus, který pro řetězec s povoleným počtem jednotlivých symbolů oddělí všechny permutace řetězce na přijímané a nepřijímané GJFA.

Kapitola 7

Implementace

Implementace problému členství JFA a GJFA byla provedena na základě objektově orientovaného paradigmatu v programovacím jazyce C++11. Byla implementována knihovna pro problém členství a jednoduchý program s textovým uživatelským rozhraním demonstrující výpočet. Bylo nutné vyřešit od sebe oddělené problémy jmenovitě grafové problémy, lexikální analýzu vstupních dat, diskrétní optimalizační problém a uložení skákajícího automatu v paměti. Pro každý typ problému je definovaná vlastní třída, řešící daný problém. Proto je možné jednotlivé třídy použít i pro jiné aplikace. Každý skákající automat před řešením musíme transformovat.

V sekci 7.1 je popsáno, jakým způsobem jsou hledány přímé cesty a elementární cykly. Sekce 7.2 popisuje způsob, jak získat závislost elementárních cyklů. V sekci 7.3 je popsáno, jak je proveden výpočet DLES. V sekci 7.4 je popsáno, jak provádíme problém členství pro GJFA. Sekce 7.5 popisuje porovnání a shrnutí dosažených výsledků.

7.1 Vyhledání přímých cest a elementárních cyklů

Pro převod skákajícího automatu na kombinaci elementárních cyklů a přímých cest budeme muset implementovat algoritmy pro nalezení elementárních cyklů a jednoduchých cest mezi dvěma uzly v grafu. V této sekci popíšeme hlavní princip těchto algoritmů.

7.1.1 Vyhledání přímých cest prohledáváním stavového prostoru

Tento algoritmus je založen na metodě zpětného vyhledávaní do hloubky stavového stromu. Metoda postupně generuje všechny jednoduché cesty mezi dvěma zvolenými uzly v grafu. Do zásobníku je vložen počáteční stav a poté jsou postupně prozkoumávány další stavy vzhledem k přechodům. Pokud je generován stav, který se již nachází na zásobníku, potom je tento stav ignorován a algoritmus pokračuje dalším pravidlem. Pokud vznikne nějaký stav $q \in F$, pak je to přímá cesta. Pokud počet všech stavů označíme jako n a pro každý stav existují pravě m pravidel, potom můžeme napsat časovou složitost jako $O(n^m)$.

7.1.2 Donald B. Johnsonův algoritmus pro nalezení všech elementárních cyklů v grafu

Efektivní algoritmus pro vyhledání všech elementárních cyklů v grafu je prezentován v [11].

Algoritmus začíná v počátečním stavu a hledá všechny elementární cykly začínající počátečním stavem postupným procházením sousedních uzlů. Algoritmus obsahuje zásobník

stavů, množinu blokovaných stavů a blokující mapu, která udává jaké všechny stavy se mají odblokovat. Aktuální stavy zkoumané jednoduché cesty jsou uchovávány na zásobníku. Po přidaní nového stavu do jednoduché cesty je tento stav zablokován, aby nemohl být použit dvakrát na té samé cestě. Pokud dalším zkoumaným stavem bude počáteční stav, potom na zásobníku je elementární cyklus a odebereme poslední stav ze zásobníku stavů a odblokujeme stav. Pokud nenarazíme na elementárních cyklus a už nejsou další sousedé k prozkoumání, potom tento stav necháme blokovaný a do blokující mapy vložíme pravidlo, že pokud se někdy tento stav odblokuje, potom se odblokuje předchozí stav. Po dokončení těchto kroků počáteční stav odstraníme z grafu a stejný postup zopakujeme pro další stavy. Algoritmus ignoruje elementární cykly, kde je jenom jeden přechod ze stavu do stejné stavu.

Časová složitost tohoto algoritmu je $O((n+e)(c+1))$ a prostorová složitost $O(n+e)$, kde n je počet uzlů, e hran a c cyklů.

7.2 Nalezení závislosti cyklů

Pro nalezení závislostí elementárních cyklů můžeme vyjádřit jako disjunkci všech povolených kombinací elementárních cyklů pro danou přímou cestu nebo můžeme snížit tento počet odstraněním nadbytečných závislostí. Vyhledávání všech závislostí cyklů je provedeno pomocí algoritmu prohledávaní stavového prostoru do hloubky. Jelikož tato operace může vyžadovat mnoho výpočetního času, před samotným vyhledáváním závislostí cyklů pro konkrétní přímou cestou je graf cyklů rozdělen na menší, silně propojené komponenty. Pro nalezení silně propojených cyklů je použit algoritmus [23]. V těch pak probíhá samotný algoritmus následovně. Do zásobníku vložíme všechny elementární cykly dosažitelné z přímé cesty a zároveň je označíme jako navštívené. Ze zásobníku vezmeme elementární cyklus a generujeme všechny jeho následníky a uložíme si tyto pravidla. Pravidla nám vyjadřují závislost mezi původním a následujícím cyklem. Generované následníky označíme jako za navštívené a pokračujeme dále. V momentě kdy jsme prošli všechny generované následníky, tak je odstraníme z navštívených. Tímto způsobem získáme všechny závislosti cyklů. Navíc nadbytečné závislosti budou odstraněny. Z těchto pravidel pak dokážeme vytvořit formuli v Presburgerově aritmetice podle lemma 37. Pro odstranění cyklické závislosti mezi cykly hledáme předcházející cykly a do výsledného pravidla je přidána konjunkce s pravidlem pro předcházející cyklus.

7.3 Výpočet lineárních diofantických rovnic

Pro optimalizační problém je implementováno několik algoritmů. Prvním algoritmem je výpočet Hermitové matice pomocí algoritmů HMM [15] a Kannan [19]. Během výpočtu Hermitovské matice provádíme stejné elementární řádkové operace i na unimodulární matici, ve které můžeme jednoduše přečíst nějaký celočíselný výsledek nebo důkaz o neřešitelnosti v celých číslech. Výhodou algoritmu HMM je, že tento výsledek je malý, respektive velikost vektoru je malá. Tudíž je pravděpodobné, že všechny jeho komponenty budou kladné. Druhý algoritmus je rychlejší pro větší matice. Protože tyto algoritmy jsou polynomiální, před aplikováním metody větvení můžeme zkusit zjistit, jestli systém má řešení v celých číslech. Tento postup se hlavně hodí pro podmnožinu JFA, která lze vyřešit v polynomiálním čase. Algoritmy můžeme aplikovat i pro JFA s vnořeným cykly, ale musíme zkontrolovat, jestli jsou splněny závislosti na vnořených cyklech.

Přesným postupem pro libovolný JFA je dekompozice Presburgerovy formule na jednotlivé ILP problémy a řešení ILP problému pomocí metody větvení a sečných nad rovin. Řešení problému je implementováno pomocí knihovny SCIP [1][14]. SCIP je volně dostupná knihovna určená pro řešení především problémů v podmínkovém celočíselném programování. SCIP obsahuje kombinaci technik pro řešení propagace domény proměnné, LP relaxace problému a metoda sečných nadrovin a analýza nepřípustných podproblémů. Všechny techniky pracují v jediném prohledávacím stromu. Nejdůležitější strukturou v SCIP jsou tzv. podmínky Každá podmínka poskytuje algoritmy Hlavním úkolem podmínek je zkontrolovat přípustnost získaného řešení vzhledem ke všem podmínkám v problému.

Další implementovanou metodou je převod problému na polynomy a vypočítaní standardní báze. Hlavním důvodem pro implementaci takového postupu bylo nalézt efektivní algoritmus pro výpočet JFA bez vnořených cyklů, ale jak se ukázalo tento postup je neefektivní. Výhodou tohoto postupu je, že převod na standardní bázi převedeme jenom jednou pro JFA. Hledání řešení pak spočívá v redukci polynomu standardní bází. Jak se ukázalo, tento postup je nepraktický a neefektivní, protože standardní báze může narůst až exponenciálně. Potom při redukce se musí procházet velké množství polynomů. Pro převod na standardní bázi je implementován algoritmus F5 [22]. Nevýhodou převodu na standardní bázi je dvojitá exponenciální časová složitost, proto i z tohoto důvodu tento postup je nepraktický. Navíc nemůžeme modelovat závislosti vnořených cyklů.

7.4 Problém členství pro obecný skákající automat

Prvním krokem bylo nutné rozdělit pravidla podle přímých cest a elementárních cyklů. Důvodem je, že řešení hledáme na jedné konkrétní cestě a na dosažitelných elementárních cyklů. Dalším důležitým krokem bylo implementování principu rychlé vyrovnávací paměti s frontou FIFO pro řetězce, které jsme již expandovali. Množství všech zkoumaných řetězců je obrovský, proto je nutné je postupně odstraňovat z asociativního pole, protože brzo by zahltili veškerou dostupnou paměť.

7.5 Testování na příkladech a výsledky

Pro účely testovaní bylo nutné implementovat generátor náhodných grafů a funkci, která generuje řetězec přijímaný obecným nebo jednoduchým skákajícím automatem. Byli implementovány dva druhy generátorů náhodných grafů. První generátor je založen na modelu Erdős-Rényi. V tomto modelu pro každou dvojici uzlů je generováno náhodné číslo v rozmezí nula až jedna, a pokud toto číslo je menší než zvolený práh, potom je vytvořena hrana. Pokud práh je právě jedna, potom jsou utvořeny hrany mezi všemi uzly. Počet všech elementárních cyklů v kompletním grafu se vypočítá jako

$$\sum_{i=1}^{n-1} \binom{n}{n-i+1}(n-i)!.$$

Počet elementárních cyklů v orientovaném grafu může s větším n exponenciálně roste. Protože skákající automat je orientovaným multigrafem, proto může existovat daleko více elementárních cyklů.

Druhý generátor náhodných grafů je založen na principu náhodného procházení. Princip je popsán následovně. Máme množinu navštívených stavů a množinu nenavštívených stavů. Z množiny nenavštívených stavů náhodně vybereme stav a zaznamenáme si hranu

z původního stavu do nově generovaného. Když jsou všechny uzly prozkoumány, označené hrany tvoří silně propojený graf. Poté v tomto grafu generujeme náhodně hrany.

U jednoduchého skákajícího automatu dokážeme efektivně vypočítat problém členství i pro vstupní řetězc s obrovským počtem znaků. Jediným omezením je paměť potřebná pro reprezentaci vektoru řešení. Výpočetní náročnost roste zejména s rostoucím počtem elementárních cyklů. Rostoucí počet elementárních cyklů a koncových stavů způsobuje značné zpomalení transformace na Presburgerovou formuli. Proto je vhodné převést skákající automat na minimalistický. Pro ověření správnosti formule bylo nutné testovat zejména případy JFA, ve kterých existují na sebe závislé cykly. Bylo nutné prověřit, jestli všechny povolené kombinace elementárních cest skutečně korespondující s vzniklou Presburgerovou formulí.

Efektivita určení problém členství vstupního řetězce do jazyka popsaného GJFA záleží na dvou věcích. Pokud neexistuje žádná variace přečtení, potom před odpovědí musíme prohledat všechny možné variace způsobů vytvoření řetězců. Při předpokladu, že každé slovo má právě dva symboly a délka vstupního řetězce je $|w| = 40$, potom maximální počet všech možných způsobů vytvoření řetězce je 7×10^{24}. Jelikož existuje obrovské množství možnosti, je pro proces přijímání implementován časovač, kdy po uplynutí zadané doby je prohledání jedné přímé cesty ukončeno a pokračuje se další přímou cestou. Ideální situace pro problém členství řetězce do jazyka popsaného GJFA je, když přímé cesty a elementární cykly jsou od sebe co nejvíce rozdílné, přesněji řečeno obsahují různé symboly a různý celkový počet jednotlivých symbolů. V tomto případě pak procházíme jen několik přímých cest a elementárních cyklů a tudíž řešení zjistíme rychleji. Je pravděpodobné, že pokud existuje řešení, potom toto řešení lze najít relativně rychle na jedné z přímých cest a přidružených elementárních cyklech. Efektivně dokážeme určit členství vstupního řetězce do jazyka popsaného GJFA vypočítat řetězec délky kolem 40 znaků.

Výpočet pomocí převodu problému na problém o polynomech je velmi neefektivní již při více jak třech proměnných. Na rozdíl od vypočítaní Hermitovské matice, kterým efektivně a jednoznačně můžeme vyřešit podmnožinu skákajících automatů řešených v polynomiálním čase.

Kapitola 8

Závěr

Seznámili jsme se s novým formálním modelem pro popis jazyků. Ukázali jsme jak převést problém členství řetězce do jazyka popsaného skákajícím automatem na diskrétní optimalizační problém. Předvedli jsme možné existující způsoby řešení optimalizačního problému a poukázali na vztah mezi skákajícím automatem a způsoby řešení. Implementovali jsme algoritmy pro vyřešení problému členství, jak jednoduchého skákajícího automatu, tak i obecného skákajícího automatu. Práce dospěla k několika důležitým poznatkům.

- Jazyk jednoduchého skákajícího automatu lze přesně matematicky vyjádřit pomocí lineárního diofantického systému popisující počet výskytu jednotlivých symbolů respektive pomocí formule Presburgerovy aritmetiky bez kvantifikátorů.
- Jednoduchý skákající automat jsme rozdělili na několik skupin jmenovitě úplně deterministický JFA, JFA bez vnořených cyklů a libovolný JFA. Úplně deterministický JFA lze vyřešit bez zpětného navrácení. Problém členství JFA bez vnořených cyklů lze vyřešit jako ILP problém. ILP problém pro pevně daný počet řádků lze vyřešit v polynomiálním čase, tudíž problém členství JFA s pevně daným počtem symbolů vstupní abecedy Σ lze vyřešit v polynomiálním čase, protože počet řádku v ILP problému koresponduje s počtem symbolů. Problém členství libovolného JFA lze vyřešit v polynomiálním navíc v případech, kdy neexistuje žádné celočíselné řešení v DLES a v dalších speciálních případech během převodu transponované unimodulární matice na Hermitovský tvar vzhledem ke původní matici DLES.
- Na základě řešení DLES jsme zjistili, že existuje podmnožina jednoduchých skákajících automatů, kterou lze vyřešit v polynomiálním čase pro libovolný vstupní řetězec. Jedinou podmínkou DLES musí potom být, že jednotlivé řádky jsou na sebe nezávislé. DLES poté můžeme vyřešit polynomiálním algoritmem převodu matice na Hermitovský tvar. Výsledkem bude jeden celočíselný bod nebo DLES nebude mít řešení. Tato podmnožina jednoduchých skákajících automatů by měla být dále zkoumána.
- Problém členství obecného skákajícího automatu lze vyřešit v polynomiálním čase jenom tehdy, když neexistuje řešení v celých číslech v korespondujícím DLES. GJFA lze redukovat na několik menších GJFA s jednou přímou cestou a tím se redukuje problém členství na to, jestli některý z těchto menších GJFA přijímá vstupní řetězec. Problém členství vstupního řetězce GJFA patří do časové třídy složitosti NP-úplná.
- Nezbytnou podmínku pro to, aby jazyk obecného skákajícího automatu byl přijímán je, aby obsahoval povolený počet symbolů vzhledem k DLES respektive vzhledem

k formuli v Presburgerově aritmetice. Ale jenom relativně malý počet permutací řetězce s povoleným počtem všech symbolů je přijímáno GJFA vzhledem ke všem permutací tohoto řetězce. Otevřeným problémem je, jestli existuje algoritmus, který pro libovolný vstupní řetězec s povoleným počtem symbolů dokáže rozlišit všechny přijímané permutace řetězce od těch nepřijímaných. Jinými slovy se ptáme na to, jestli existuje nějaká charakteristika jazyka popsaného GJFA.

- Jednoduchý skákající automat lze vyjádřit vzhledem k Presburgerově formuli jako konečný automat s n čtecími hlavami. Otázka členství pak může být formulována to, jestli jestli jazyk konečného automatu s více čtecími hlavami není prázdný. Navíc konečným automatem s více čtecími hlavami můžeme vyjádřit všechny přijímané kombinace cyklů. Potom na problém můžeme nahlížet na to, jestli jednotlivé výskyty symbolů jsou zároveň přijímány m čtecími hlavami.

Literatura

[1] Achterberg, T.: SCIP: solving constraint integer programs. *Mathematical Programming Computation*, ročník 1, č. 1, 2009: s. 1–41, ISSN 1867-2957.

[2] Boudet, A.; Comon, H.: *Diophantine equations, Presburger arithmetic and finite automata*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, ISBN 978-3-540-49944-2, 30–43 s.

[3] Church, A.: *An Unsolvable Problem of Elementary Number Theory*. American Journal of Mathematics, 1936.

[4] Cooper, D. C.: Theorem proving in arithmetic without multiplication. *Machine Intelligence 7*, 1972: s. 91–99.
URL http://citeseerx.ist.psu.edu/showciting?cid=697241

[5] De Loera, J. A.: The many aspects of counting lattice points in polytopes. *Mathematische Semesterberichte*, ročník 52, č. 2, 2005: s. 175–195, ISSN 1432-1815.

[6] Fernández, M.: *Models of Computation An introduction to Computability theory*. American Journal of Mathematics, 2009, ISBN 978-1-84882-433-1.

[7] Fischer, M. O., Michael J.and Rabin: *Super-Exponential Complexity of Presburger Arithmetic*. Vienna: Springer Vienna, 1998, ISBN 978-3-7091-9459-1, 122–135 s.

[8] Gershenfeld, N.; Chuang, I. L.: *Quantum Computing with Molecules*. [Online; navštíveno 1.05.2017].
URL http://cba.mit.edu/docs/papers/98.06.sciqc.pdf

[9] H. W. Lenstra, J.: *Integer Programming with a Fixed Number of Variables*, ročník 8. Mathematics of Operations Research, 1983.

[10] Hilbert, D.; Ackermann, W.: *Principles of Mathematical Logic* . American Mathematical Society, 1999, ISBN 0821820249.

[11] Johnson, D. B.: Finding All the Elementary Circuits of a Directed Graph. *SIAM J. Comput.*, ročník 4, č. 1, 1975: s. 77–84.

[12] Křivka, Z.; Meduna, A.: Jumping Grammars. *International Journal of Foundations of Computer Science*, ročník 26, č. 06, 2015: s. 709–731.
URL http://www.worldscientific.com/doi/abs/10.1142/S0129054115500409

[13] Loera, J. A. D.; Hemmecke, R.; Tauzer, J.; aj.: Effective lattice point counting in rational convex polytopes. *Journal of Symbolic Computation*, ročník 38, č. 4, 2004: s.

1273 – 1302, ISSN 0747-7171, symbolic Computation in Algebra and Geometry.
URL http://www.sciencedirect.com/science/article/pii/S0747717104000422

[14] Maher, S. J.; Fischer, T.; Gally, T.; aj.: The SCIP Optimization Suite 4.0. 2017.

[15] Majeswki, G. H. B. S.; Matthews, K. R.: *Extended GCD and Hermite Normal Form Algoritmhs via Lattice*. A K Peters Ltd, 1998, ISBN ISBN 0-8218-3804-0.

[16] Matthews, K.: Solving AX = B using the Hermite normal form. [Online; navštíveno 1.05.2017].
URL http://www.numbertheory.org/pdfs/ax=b.pdf

[17] Meduna, A.; Zemek, P.: *Chapter 17 Jumping Finite Automata*. New York, NY: Springer New York, 2014, ISBN 978-1-4939-0369-6, s. 567–585.

[18] a Philippe Loustaunau, W. W. A.: *An Introduction to Gröbner basis*. American Mathematical Society, 2000, ISBN ISBN 0-8218-3804-0.

[19] Ravindran Kannan, A. B.: *Polynomial Algorithms for Computing the Smith and Hermite Normal Forms of an Integer Matrix*. SIAM J. COMPUT., 1979, ISBN ISBN 0-8218-3804-0.

[20] Schrijver, A.: *Theory of Linear and Integer Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1986, ISBN 0-471-90854-1.

[21] Stansifer, R.: Presburger"s Article on Integer Arithmetic: Remarks and Translation. Technická zpráva, Ithaca, NY, USA, 1984.

[22] Stegers, T.: Faugere's F5 Algorithm Revisited. Cryptology ePrint Archive, Report 2006/404, 2006, http://eprint.iacr.org/2006/404.

[23] Tarjan, R.: Depth first search and linear graph algorithms. *SIAM JOURNAL ON COMPUTING*, ročník 1, č. 2, 1972.

[24] Tomás, A. P.; Filgueiras, M.: *An Algorithm for Solving Systems of Linear Diophantine Equations in Naturals*. SIAM J.Comput, 1975.

[25] Turing, A. M.: On Computable Numbers with an Application to the Entscheidungsproblem. [Online; navštíveno 1.05.2017].
URL https://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf

[26] Wolper, P.; Boigelot, B.: *An automata-theoretic approach to Presburger arithmetic constraints*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, ISBN 978-3-540-45050-4, 21–32 s.

Přílohy

Příloha A

Manuál pro implementovanou knihovnu skákající automaty

V této kapitole popíšeme postup přeložení knihovny pro skájající automaty. Knihovna používá knihovny Boost zejména pro aritmetiku s libovolnou přesností, framework SCIP pro definování a výpočet Presburgerově formuli a v neposlední řadě standardní knihovnu jazyka C++ pro základní datové typy jako např. hashovací tabulka nebo vyhledávací strom. Dokumentace jednotlivých implementovaných tříd a jejich rozhraní je provedena pomocí programu doxygen. Pro určení problému členství máme dvě možnosti. Můžeme si pomocí této knihovny implementovat vlastní program nebo použít interaktivní program implementovaný společně s knihovnou. Tento program dokáže řešit problém členství vstupního řetězce pro libovolný skákající automat.

A.1 Překlad zdrojových kódů knihovny a dokumentace

Pro překlad obsahu přiloženého média je nutné mít nainstalovaný kompilátor pro jazyk C++ s podporou mezinárodního standardu C++11. Pro překlad potom stačí zadat příkaz `make` v kořenovém adresáři. Pokud vše proběhne v pořádku, měla by se objevit na konzolové řádce zpráva o úspěšném překladu.

Pokud se tak nestane, tak nejčastějším problémem bude, že váš systém neobsahuje knihovnu GMP pro aritmetiku s libovolnou přesností, knihovnu Readline pro editaci konzolové řádky nebo knihovnu ZIMPL pro popis matematických formulí pro SCIP. Řešením tohoto problému je při překladu deaktivovat tyto knihovny následujícím způsobem.

```
make READLINE=false GMP=false ZIMPL=false
```

Pro vygenerování webové dokumentace pro všechny funkce, třídy a datové typy je nutné zadat příkaz

```
make doxygen.
```

V této dokumentaci, která se po vygenerování dokumentace bude nacházet v adresáři `./doc`, najdete detailní popis implementovaných algoritmů a datových struktur.

A.2 Popis formátu skákajících automatů

Formát popisující skákající automat je definován následovně. Řetězec je libovolná sekvence znaků a písmeno je libovolný symbol.

```
<jfm> ::= ({<stavy>},{<symboly>},{<pravidla>},<stav> {<stavy>})
<stavy> ::= <řetězec> | <stavy> , <řetězec>
<symboly> ::= <písmeno> | <symboly> , <písmeno>
<pravidla> ::= <pravidlo> | <pravidla> , <pravidlo>
<pravidla> ::= <pravidlo> | <pravidla> , <pravidlo>
<pravidlo> ::= <řetězec> '<řetězec>' -> <řetězec>
```

Povoleným formátem je i když jsou některé množiny prázdne, ale musí být minimálně dodržena podmínka, že daný stav nebo symbol se vyskytuje v množině stavů nebo vstupní abecedě.Příkladem akceptovaného skákajícího automatu je například následující popis.

```
({s0,q1,q2,q3},a,b,c,s0 'aab' -> q1,q1 'bbc' -> q2,s0,{q3,q2})
```

A.3 Popis použití a příkazů pro program implementující problém členství

Po provedení překladu je možné spustit hlavní program zadaním následujícího příkazu do konzole.

```
./bin/jfm
```

Zde je ukázka nejdůležitějších funkcí pro práci s programem určující problém členství vstupního řetězce skákajícím automatem. Pro úplný výčet všech funkcí stačí zadat příkaz help nebo přečíst projektovou dokumentaci.

`read file`

pokusí se přečíst skákající automat podle relativní cesty k souboru, která je zadána jako první argument

na standardní výstup vypíše, jestli se přečtení souboru podařilo nebo ne

`transform`

nalezne všechny elementární cykly a přímé cesty

nalezene všechny propojené elementární cykly a přímé cesty a mezi elementárními cykly najde všechny závislosti a následně převede skákající automat na diskrétní optimilizační problém definovaný v Presburgově aritmetice

`random alphabet wordsize statessize`

generuje nový skákající automat se silně propojenýma komponentama

prvním argumentem je abeceda, zadaná jakou souvislý text

druhým argumentem je maximální délka generovaného pravidla

třetím argumentem je maximální počet stavů

před generovaním je nutné se ujistit, že současný automat není již definovaný

`generate cyclesize`

generuje řetězec, který je přijíman skákajícím automatem

argument cyclesize definuje maximalní počet generovaných cyklů pro nějaký cyklus

před zadáním příkázu solve je nutné skákající automat transformovat, protože generování slova probíhá na základě náhodné vkládání slov podle vzhledem k přímé cestě a cyklů

vygenerovaný řetězec je uložen

`print`

vytiskne skákající automat na standardní výstup

`solve input`

vypočítá jestli je daný řetězec přijímán skákajícím automatem

použije nejefektivnější algoritmus pro určení problému členství podle toho, jestli je právě definován JFA nebo GJFA

před zadáním příkázu solve je nutné skákající automat transformovat

pokud je zadán agument jako random, potom výpočet bude probíhat pro právě uložený náhodny řetězec generovaný příkazem generate

Před samotným řešením řetězce je nutné skákající automat transformovat na formuli v Presburgově arimetice. Veškeré testy i s vysvětlením se nacházejí v adresáři `./data`. Typická sekvence pro vyřešení problému členství skákajícího automatu pak za předpokladu, že se nacházíme v kořenovém adresáři, vypadá následovně.

```
./bin/jfm ./data/jfa1

transform

solve abcd
```