VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

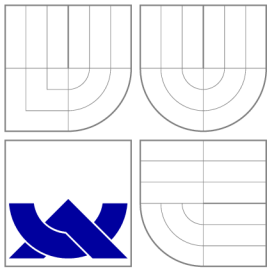
MODIFIKOVANÉ ZÁSOBNÍKOVÉ AUTOMATY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

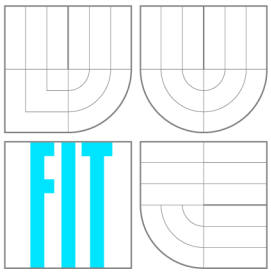
AUTOR PRÁCE
AUTHOR

DAVID HROMÁDKA

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODIFIKOVANÉ ZÁSOBNÍKOVÉ AUTOMATY

MODIFIED PUSHDOWN AUTOMATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — DAVID HROMÁDKA
AUTHOR

VEDOUCÍ PRÁCE — prof. RNDr. ALEXANDER MEDUNA, CSc.
SUPERVISOR

BRNO 2011

Abstrakt

Tato práce zavádí *omezené Hromádkovy automaty* jako rozšíření zásobníkových automatů. Toto rozšíření spočívá v tom, že automat může za běhu vytvářet nové zásobníky, vkládat do nich symboly vstupní abecedy, spojovat je a porovnávat se vstupním řetězcem. Počet takto vytvořených zásobníků je omezen zvolenou konstantou n. Dále tato práce popisuje implementaci aplikace, která činnost těchto automatů realizuje a hledá posloupnost derivačních kroků, kterými určitý omezený Hromádkův automat přijme zadaný vstupní řetězec.

Abstract

This work introduces *limited Hromádka's automata* as an extension of classical pushdown automata. This extension means that the automaton is able to create new pushdowns, insert input symbols into them, join them and compare with the input string in run time. The number of such pushdowns is limited by choosen constant n. This work also describes the implementation of the application, that realizes the activity of this automata and is looking for the sequence of derivational steps, by which the automaton accepts the input string.

Klíčová slova

formální jazyk, automat, zásobník, zásobníkový automat, čistý zásobník

Keywords

formal language, automaton, pushdown, pushdown automaton, pure pushdown

Citace

David Hromádka: Modifikované zásobníkové automaty, bakalářská práce, Brno, FIT VUT v Brně, 2011

Modifikované zásobníkové automaty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. RNDr. Alexandra Meduny, CSc.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
David Hromádka
27. dubna 2011

Poděkování

Chtěl bych poděkovat prof. RNDr. Alexandru Medunovi, CSc., který tuto bakalářskou práci vedl a konzultoval. Pomohl mi s osnovou práce a poradil literaturu vhodnou k nastudování.

© David Hromádka, 2011.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **2**

2 Definice **5**
2.1 Doplňující definice 5
2.2 Derivační krok 6
2.3 Přijímaný jazyk 6
2.4 Ilustrace definice 7
2.5 Neformální popis 9
2.5.1 Konfigurace 9
2.5.2 Pravidla 9

3 Použitý algoritmus **11**
3.1 Komentáře k algoritmu 13
3.2 Proč je nutno velikost zásobníku operací limitovat? 14
3.3 Možné cesty dalšího vývoje 14
3.3.1 Upřednostnění porovnání 14
3.3.2 Nalezení nejkratší posloupnosti 14

4 Příklady **15**

5 Implementace **18**
5.1 Postup implementace 18
5.2 Lexikální analýza 19
5.3 Syntaktická a sémantická analýza 21

6 Závěr **23**
6.1 O reprezentaci automatu 24
6.2 Zhodnocení dosažených výsledků 24
6.3 Možný budoucí vývoj projektu 24

Kapitola 1

Úvod

V počátcích informatiky existovala potřeba vyjádřit počítačový program ve formě srozumitelné počítači i člověku. Protože počítač rozumí v podstatě jen posloupnostem nul a jedniček, bylo řešení tohoto problému náročné a zaměstnalo nemálo inženýrů. Výsledkem této snahy byl vznik programovacích jazyků druhé a později i třetí a čtvrté generace, na počátku zejména jazyka symbolických instrukcí.

Aby byl programovací jazyk v praxi použitelný, musí být přesně definován. Jazyk, který je přesně matematicky definován, nazýváme jazykem formálním. Všechny dnes běžně používané programovací jazyky jsou formální.

Matematicky přesnou definicí syntaxe těchto jazyků se zabývá obor teoretické informatiky zvaný *teorie formálních jazyků*. Tento obor si klade za cíl společným (univerzálním) a abstraktním způsobem exaktně popsat (specifikovat) všechny potřebné formální jazyky. Základním předpokladem je zde schopnost pomocí konečného popisu specifikovat nekonečný jazyk. (Příkladem takového nástroje jsou regulární výrazy.)

Pro popis syntaxe formálních jazyků se v teorii formálních jazyků ustálily dva základní typy nástrojů – gramatiky a automaty (teoretické výpočetní modely). Gramatika se vyznačuje přítomností terminálních a neterminálních symbolů a přepisovacích pravidel, podle kterých se části řetězce přepisují jinými. Automat se vyznačuje přítomností konečného stavového řízení a přechodových pravidel, která umožňují přechod z jednoho stavu do jiného. Mezi těmito oblastmi existují přesahy; příkladem jsou stavové gramatiky, které při přepisování používají stavového řízení. Existují různé formy gramatik a různé formy automatů. Některé gramatiky dovolují popsat širší třídu jazyků než jiné. Podle [1] existuje ke každému typu gramatiky podle Chomského hierarchie automat odpovídající výpočetní síly, tedy výpočetní model takový, který dokáže přijímat právě ty jazyky, které daná gramatika dokáže generovat.

Formální jazyky a s nimi i gramatiky a automaty nalézají využití v samotné informatice, ale i mimo ni, například v bioinformatice. V rámci informatiky je nejvýznamnějším použitím teorie formálních jazyků popis programovacích jazyků a konstrukce jejich překladačů. Obvyklým postupem je napsat gramatiku programovacího jazyka (pokud již nebyla napsána), převést ji na příslušný automat, ten přizpůsobit potřebám praxe a implementovat.

Jak se píše v [5], překladač programovacího jazyka se typicky skládá ze šesti částí – lexikálního analyzátoru, syntaktického analyzátoru, sémantického analyzátoru, generátoru vnitřního kódu, optimalizátoru a generátoru cílového kódu.

Úlohou lexikálního analyzátoru je rozdělit zdrojový kód na lexikální jednotky nazývané *tokeny*. Příkladem takového tokenu může být číslo 123 nebo identifikátor `TOK_HASH`. Podoby

konkrétních typů tokenů bývají popsány regulárním výrazem a jejich rozpoznávání bývá implementováno konečným stavovým automatem.

Syntaktický analyzátor přijímá od lexikálního analyzátoru posloupnost tokenů a kontroluje, zda tato posloupnost reprezentuje syntakticky správně napsaný program, tedy zda její stavba odpovídá gramatice daného jazyka. Obvykle je jazyk popsán bezkontextovou gramatikou a syntaktický analyzátor je implementován jako zásobníkový automat.

Z výše uvedeného vyplývá, že zásobníkové automaty a s nimi související bezkontextové gramatiky jsou významnými nástroji nejen v teorii formálních jazyků, ale i v související praxi. Jsou používány autory kompilátorů většiny programovacích jazyků, a dokonce již vznikly i automatizované nástoje, které se zabývají touto problematikou bez zásahu člověka (např. program *yacc*).

V průběhu historie byla vytvořena řada modifikací zásobníkových automatů, jejichž cílem bylo zvýšit nebo naopak snížit jejich vyjadřovací sílu. Pokud jde o začátečníky v oboru formálních jazyků, většina jejich pokusů zaměřených tímto směrem skončí formálním modelem s výpočetní silou Turingova stroje. Příkladem může být dvouzásobníkový automat (viz [7]). Tomu však lze zabránit stanovením vhodných omezení v činnosti daného modelu.

Tato bakalářská práce zavádí *omezený Hromádkův automat* jako novou modifikaci klasického zásobníkového automatu. Klíčovým prvkem této modifikace je to, že omezený Hromádkův automat může vedle základního zásobníku používat i další zásobníky. Tyto zásobníky jsou ovšem tzv. čisté zásobníky; mohou obsahovat pouze symboly vstupní abecedy a nelze je z nich snadno vyjmout.

Vyjadřovací síla omezeného Hromádkova automatu zásadně závisí na parametru n, který stanovuje horní mez počtu zásobníků v konfiguraci automatu. Pro $n = 1$ je jeho vyjadřovací síla stejná jako u zásobníkového automatu, protože s tímto omezením jde o obyčejný zásobníkový automat s lehce odlišnou syntaxí a pohodlnější detekcí dna zásobníku. Pro jakékoliv vyšší n ale automat získává na síle, protože přítomnost více zásobníků mu umožňuje vkládat podřetězce na více míst v očekávaném řetězci, a tak provádět např. expanzi neterminálů odpovídající gramatiky hlouběji v zásobníku, podobně jako hluboký zásobníkový automat zavedený v [4].

Symbol jednou vložený do čistého zásobníku ale automat nemůže snadno vyjmout ani číst. Jedinou možností, jak toto realizovat, je spojit čisté zásobníky s hlavním zásobníkem, a po provedení operací s vedlejšími zásobníky je obnovit operací odříznutí $\otimes$. Tímto se ovšem na dno každého zúčastněného čistého zásobníku dostane neprázdný řetězec vstupních symbolů, kterým byly zásobníky spojeny.

Základem omezeného Hromádkova automatu je zásobníkový automat se všemi jeho možnostmi. Podrobnější informace o klasické zásobníkovém automatu lze nalézt v [3]. K němu jsou dodány některé nové operace umožňující vytváření nových čistých zásobníků, vkládání vstupních symbolů do nich, jejich spojování a načítání znaků vstupního řetězce porovnáním s vrcholem zásobníku s nejvyšším číselným označením.

Omezený Hromádkův automat může být použit v syntaktickém analyzátoru pro analýzu struktur, které nejsou bezkontextové. Jeho vyjadřovací síla je vyšší než u běžného zásobníkového automatu, což je zřejmé z některých příkladů uvedených v kapitole 4. To umožňuje použít v daném programovacím jazyce konstrukce, které by nebylo možno rozpoznat běžným zásobníkovým automatem, ačkoliv použití dvojzásobníkového automatu nebo jiné obdoby Turingova stroje by bylo možná jednodušším řešením.

Praktické použití omezeného Hromádkova automatu je omezené, a to zejména tím, že je z principu nedeterministický. Součástí zadání bylo implementovat aplikaci omezeného Hromádkova automatu v oblasti syntaktické analýzy. Implementovaná aplikace přijímá ze

standardního vstupu definici omezeného Hromádkova automatu a jeho vstupního řetězce v příslušném formátu a na standardní výstup odpovídá, zda automat takový řetězec přijme. Problém nedeterminismu byl vyřešen použitím metod umělé inteligence, jak je podrobněji popsáno v kapitole 5. Je-li nastavená hloubka prohledávání nižší, než je délka nejkratší posloupnosti derivačních kroků, kterou automat vstupní řetězec přijme, aplikace selže a řetězec zamítne. Je-li však nastavená hloubka prohledávání příliš vysoká, může trvat prohledávání stavového prostoru velmi dlouho. Správná funkce aplikace je tedy závislá na vhodně zvolené hloubce prohledávání.

V kapitole 2 je omezený Hromádkův automat definován. Na začátku kapitoly je uvedena jeho formální definice včetně všech doplňujících definic, je definován derivační krok a jazyk přijímaný omezeným Hromádkovým automatem. Následuje podrobná ukázka praktické aplikace této definice a neformální popis, který formální definici automatu doplňuje, blíže vysvětluje a činí přehlednější.

Kapitola 3 popisuje algoritmus prohledávání stavového prostoru, který tvoří funkční jádro aplikace realizující kroky daného automatu. Protože omezený Hromádkův automat musí být nutně nedeterministický, není tento algoritmus triviální, a proto je mu věnována celá tato kapitola. Algoritmus je nejprve uveden a následně okomentován.

Kapitola 4 obsahuje příklady jazyků, které může omezený Hromádkův automat přijímat, a jim odpovídajících automatů, které je přijímají. Alespoň jeden uvedený jazyk není bezkontextový.

Kapitola 5 se věnuje podrobnému popisu implementace aplikace, která je předmětem této práce. Je v ní popsána zejména lexikální a syntaktická analýza, jejíž implementace byla nezbytná kvůli načítání specifikace automatu a vstupního řetězce ze souboru.

Závěr obsahuje zhodnocení práce, přehled, čeho bylo dosaženo, a přehled možných cest budoucího vývoje této tématiky.

Kapitola 2

Definice

Omezený Hromádkův automat je definován jako uspořádaná sedmice

$$A = (Q, \Sigma, Z, R, k, F, n), \text{ kde:}$$

- Q je abeceda stavů,
- Σ je vstupní abeceda (abeceda teriminálů),
- Z je abeceda neterminálů,
- $R \subseteq P$ je konečná podmnožina množiny pravidel,
- $k \in L_k$ je počáteční vnitřní konfigurace daného automatu,
- $F \subseteq L_k$ je konečná množina koncových vnitřních konfigurací,
- a $n \geq 1$ je maximální počet zásobníků.

2.1 Doplňující definice

- $A_K = \Sigma \cup Z \cup \{\#_i \mid 0 \leq i \leq n\}$ je redukovaná konfigurační abeceda daného automatu (konfigurační abeceda bez stavů), přičemž platí: $|A_K \cup Q| = |Q| + |\Sigma| + |Z| + n + 1$ (vynucuje, aby příslušné abecedy byly po párech disjunktní).
- $A_C = \Sigma \cup Z \cup \{\Omega, \Upsilon\}$ je podmínková abeceda daného automatu, přičemž platí: $|A_C| = |\Sigma| + |Z| + 2$ (vynucuje disjunktnost abeced). Ω znamená podmínku prázdnosti, Υ znamená provedení bez podmínky (v každém případě).
- $l(i) : \mathbb{N} \to 2^{A_K^*}$ je zobrazení definované rekurzívně takto ($\mathbb{N}$ značí množinu přirozených čísel):

$$\begin{aligned} l(1) &= \{\#_1\}.(\Sigma \cup Z)^*.\{\#_0\} \\ l(i) &= \{\#_i\}.\Sigma^*.l(i-1) \qquad \text{pro } i > 1 \end{aligned}$$

- $L_k = Q.\bigcup_{i=1}^{n} l(i)$ je množina vnitřních konfigurací daného automatu,
- $L_K = L_k.\Sigma^*$ je množina konfigurací daného automatu.

1	$m = p_1 c \#_0 u p_2$	$k_1 = p_1 r_1 \#_1 c r_2 t$ $k_2 = p_2 r_1 \#_1 u r_2 t$	$c \in \Sigma \cup Z, u \in (\Sigma \cup Z)^*$
2	$m = p_1 c \Upsilon \#_0 u p_2$	$k_1 = p_1 r_1 \#_1 r_2 t$ $k_2 = p_2 r_1 \#_1 u r_2 t$	$u \in (\Sigma \cup Z)^*$
3	$m = p_1 c \Omega \#_0 u p_2$	$k_1 = p_1 r_1 \#_1 \#_0 t$ $k_2 = p_2 r_1 \#_1 u \#_0 t$	$u \in (\Sigma \cup Z)^*$
4	$m = p_1 \#_i u p_2$	$k_1 = p_1 r_1 \#_{i+1} r_2 t$ $k_2 = p_2 r_1 \#_{i+1} u r_2 t$	$0 < i < n, u \in \Sigma^*$
5	$m = p_1 \bigtriangledown p_2$	$k_1 = p_1 \#_i a r_2 a t$ $k_2 = p_2 \#_i r_2 t$	$0 < i \leq n, a \in \Sigma$
6	$m = p_1 \otimes p_2$	$k_1 = p_1 r_1 \#_1 u v \#_0 t$ $k_2 = p_2 f(r_1) \#_2 u \#_1 v \#_0 t$	$u \in \Sigma^*, v \in \{\varepsilon\} \cup Z.(\Sigma \cup Z)^*$
7	$m = p_1 \oplus_i v p_2$	$k_1 = p_1 f(r_1) \#_{i+1} u \#_i w \#_{i-1} r_2 t$ $k_2 = p_2 r_1 \#_i u v w \#_{i-1} r_2 t$	$u \in \Sigma^*, v \in \Sigma.\Sigma^*, w \in (\Sigma \cup Z)^*$

Tabulka 2.1: Třídy pravidel

- $P = Q.(A_C.\{\#_0\}.(\Sigma \cup Z)^* \cup \{\#_i \mid 0 < i < n\}.\Sigma^* \cup$
  $\cup \{\oplus_i \mid 0 < i < n\}.\Sigma.\Sigma^* \cup \{\bigtriangledown, \otimes\}).Q$ je množina všech pravidel možných v daném automatu.

2.2 Derivační krok

Nechť pravidlo $m \in R$ a platí podmínky uvedené ve všech sloupcích kromě prvního na některém řádku tabulky 2.1, $k_1, k_2 \in L_K$, $t \in \Sigma^*$, $r_1 \in A_K^*$, $r_2 \in \{\varepsilon\} \cup A_K^*.\{\#_0\}$ a $p_1, p_2 \in Q$. Pak k_1 přímo derivuje k_2 podle pravidla m a píšeme $k_1 \vdash k_2\ [m]$ nebo zjednodušeně $k_1 \vdash k_2$.

Relaci $\vdash \subseteq A_K^* \times A_K^*$ nazýváme *přechodovou relací* a definujeme její tranzitivní uzávěr $\vdash^+$ a tranzitivní a reflexivní uzávěr „derivuje“ $\vdash^*$. Vedle toho můžeme ještě definovat relaci $\vdash^o$ „derivuje v o krocích“ následovně:

$$\vdash^o = \begin{cases} \{(k_1, k_1) \mid k_1 \in A_K^*\} & \text{pro } o = 0 \\ \vdash & \text{pro } o = 1 \\ \{(k_1, k_2) \mid k_1 \vdash k' \vdash^{o-1} k_2\} & \text{jinak} \end{cases}$$

2.3 Přijímaný jazyk

Jazyk $L(A)$ přijímaný omezeným Hromádkovým automatem A je definován takto:

$$L(A) = \{w \in \Sigma^* \mid \exists f \in F, k.w \vdash^* f\}$$

Jedná se tedy o množinu řetězců nad vstupní abecedou takových, že konfigurace vzniklá připojením takového řetězce za počáteční vnitřní konfiguraci derivuje některou koncovou vnitřní konfiguraci. Vyžaduje se tedy kompletní načtení vstupního řetězce. Vyprázdnění zásobníků není vyžadováno, protože koncové konfigurace mohou obsahovat v zásobnících různé symboly, dokonce mohou v hlavním zásobníku zůstat i neterminály. Množina F je však vždy konečná, takže i koncových konfigurací je jen konečně mnoho. Pokud je F prázdná, pak automat nepřijme žádný řetězec.

2.4 Ilustrace definice

Nyní si uvedeme příklad omezeného Hromádkova automatu a ilustrujeme na něm výše uvedenou formální definici. Mějme omezený Hromádkův automat $A = (Q, \Sigma, Z, R, k, F, n)$ definovaný takto:

$$
\begin{aligned}
Q &= \{s_1, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}, s_{12}, s_{13}, s_{14}, s_{15}, s_{16}, s_{17}, s_{18}, s_{19}\} \\
\Sigma &= \{a, b, c\} \\
Z &= \{M, N, O\} \\
R &= \{s_1 \otimes s_1, s_1 \otimes s_3, s_3 \#_2 a s_4, s_3 \#_2 b s_6, s_4 \Upsilon \#_0 M s_5, s_5 \triangledown s_3, s_6 \triangledown s_7, s_7 \#_2 b s_8, \\
&\quad s_7 M \#_0 s_{10}, s_8 \triangledown s_7, s_9 \#_1 a s_{10}, s_{10} \#_2 c s_{11}, s_{11} M \#_0 s_9, s_{11} N \#_0 s_{12}, s_{12} \oplus_2 a s_{13}, \\
&\quad s_{13} \triangledown s_{14}, s_{13} \oplus_1 b s_{16}, s_{14} \triangledown s_{15}, s_{15} \Upsilon \#_0 c s_{13}, s_{16} \Upsilon \#_0 b s_{16}, s_{16} \triangledown s_{18}, s_{18} \triangledown s_{18}, \\
&\quad s_{18} \Upsilon \#_0 O s_{19}\} \\
k &= s_1 \#_1 NO \#_0 \\
F &= \{s_1 \#_2 \#_1 NO \#_0, s_{19} \#_1 OO \#_0\} \\
n &= 3
\end{aligned}
$$

Na tomto automatu si ukážeme praktický význam doplňujících definic a příklady derivačních kroků. Přijímá jazyk $L(A) = \{\varepsilon, a^n b^m c^n a^n b^o c^n \mid n \geq 0, m, o > 0\}$.

Redukovaná konfigurační abeceda $A_K = \{a, b, c, M, N, O, \#_0, \#_1, \#_2\}$, podmínková abeceda $A_C = \{a, b, c, M, N, O, \Omega, \Upsilon\}$, zobrazení $l(i)$ pro $0 < i \leq n$:

$$l(1) = \{\#_1\}.(\Sigma \cup Z)^*\{\#_0\} \tag{2.1}$$

$$l(2) = \{\#_2\}.\Sigma^*.\{\#_1\}.(\Sigma \cup Z)^*\{\#_0\} \tag{2.2}$$

$$l(3) = \{\#_3\}.\Sigma^*.\{\#_2\}.\Sigma^*.\{\#_1\}.(\Sigma \cup Z)^*\{\#_0\} \tag{2.3}$$

Množina všech vnitřních konfigurací $L_k = l(1) \cup l(2) \cup l(3)$. Hodnota $l(i)$ pro určité i představuje množinu všech vnitřních konfigurací daného automatu s právě i zásobníky. Protože $n = 3$, automat může mít jeden až tři zásobníky (aktuální počet zásobníků se může derivačními kroky automatu měnit), takže množina všech vnitřních konfigurací L_k zahrnuje všechny tyto hodnoty zobrazení l.

Množina všech konfigurací $L_K = L_k.\Sigma^*$. Množinu všech možných pravidel nebudu rozebírat.

Prázdný řetězec automat přijme v jednom derivačním kroku:

$$s_1 \#_1 NO \#_0 \vdash s_1 \#_2 \#_1 NO \#_0 \, [s_1 \otimes s_1] \tag{2.4}$$

Nechť vstupním řetězcem automatu je *aabbbccaabcc*. Tento řetězec již vede na celkem dlouhou posloupnost derivačních kroků:

$$s_1 \#_1 NO \#_0 aabbbccaabcc \vdash s_1 \#_2 \#_1 NO \#_0 aabbbccaabcc \, [s_1 \otimes s_1] \tag{2.5}$$

$$\ldots \vdash s_3 \#_3 \#_2 \#_1 NO \#_0 aabbbccaabcc \, [s_1 \otimes s_3] \tag{2.6}$$

Automat nejprve spočítá znaky a na vstupu a jejich počet si uloží do hlavního zásobníku ve formě neterminálů M:

$$\ldots \vdash s_4 \#_3 a \#_2 \#_1 NO \#_0 aabbbccaabcc \, [s_3 \#_2 a s_4] \tag{2.7}$$

$$\ldots \vdash s_5 \#_3 a \#_2 \#_1 MNO \#_0 aabbbccaabcc \, [s_4 \Upsilon \#_0 M s_5] \tag{2.8}$$

$$\ldots \vdash s_3\#_3\#_2\#_1 MNO\#_0 abbbccaabcc\ [s_5 \triangledown s_3] \tag{2.9}$$

$$\ldots \vdash s_4\#_3 a\#_2\#_1 MNO\#_0 abbbccaabcc\ [s_3\#_2 a s_4] \tag{2.10}$$

$$\ldots \vdash s_5\#_3 a\#_2\#_1 MMNO\#_0 abbbccaabcc\ [s_4 \Upsilon\#_0 M s_5] \tag{2.11}$$

$$\ldots \vdash s_3\#_3\#_2\#_1 MMNO\#_0 bbbccaabcc\ [s_5 \triangledown s_3] \tag{2.12}$$

Následně automat pohltí všechny znaky b, dokud nenarazí na c:

$$\ldots \vdash s_6\#_3 b\#_2\#_1 MMNO\#_0 bbbccaabcc\ [s_3\#_2 b s_6] \tag{2.13}$$

$$\ldots \vdash s_7\#_3\#_2\#_1 MMNO\#_0 bbccaabcc\ [s_6 \triangledown s_7] \tag{2.14}$$

$$\ldots \vdash s_8\#_3 b\#_2\#_1 MMNO\#_0 bbccaabcc\ [s_7\#_2 b s_8] \tag{2.15}$$

$$\ldots \vdash s_7\#_3\#_2\#_1 MMNO\#_0 bccaabcc\ [s_8 \triangledown s_7] \tag{2.16}$$

$$\ldots \vdash s_8\#_3 b\#_2\#_1 MMNO\#_0 bccaabcc\ [s_7\#_2 b s_8] \tag{2.17}$$

$$\ldots \vdash s_7\#_3\#_2\#_1 MMNO\#_0 ccaabcc\ [s_8 \triangledown s_7] \tag{2.18}$$

Použitím zásobníků 1 a 2 automat vygeneruje $c^n a^n$ podle uložených neterminálů M:

$$\ldots \vdash s_{10}\#_3\#_2\#_1 MNO\#_0 ccaabcc\ [s_7 M\#_0 s_{10}] \tag{2.19}$$

$$\ldots \vdash s_{11}\#_3 c\#_2\#_1 MNO\#_0 ccaabcc\ [s_{10}\#_2 c s_{11}] \tag{2.20}$$

$$\ldots \vdash s_9\#_3 c\#_2\#_1 NO\#_0 ccaabcc\ [s_{11} M\#_0 s_9] \tag{2.21}$$

$$\ldots \vdash s_{10}\#_3 c\#_2 a\#_1 NO\#_0 ccaabcc\ [s_9\#_1 a s_{10}] \tag{2.22}$$

$$\ldots \vdash s_{11}\#_3 cc\#_2 a\#_1 NO\#_0 ccaabcc\ [s_{10}\#_2 c s_{11}] \tag{2.23}$$

$$\ldots \vdash s_{12}\#_3 cc\#_2 a\#_1 O\#_0 ccaabcc\ [s_{11} N\#_0 s_{12}] \tag{2.24}$$

$$\ldots \vdash s_{13}\#_2 ccaa\#_1 O\#_0 ccaabcc\ [s_{12} \oplus_2 a s_{13}] \tag{2.25}$$

Následuje fáze porovnávání:

$$\ldots \vdash s_{14}\#_2 caa\#_1 O\#_0 caabcc\ [s_{13} \triangledown s_{14}] \tag{2.26}$$

$$\ldots \vdash s_{15}\#_2 aa\#_1 O\#_0 aabcc\ [s_{14} \triangledown s_{15}] \tag{2.27}$$

$$\ldots \vdash s_{13}\#_2 aa\#_1 cO\#_0 aabcc\ [s_{15} \Upsilon\#_0 c s_{13}] \tag{2.28}$$

$$\ldots \vdash s_{14}\#_2 a\#_1 cO\#_0 abcc\ [s_{13} \triangledown s_{14}] \tag{2.29}$$

$$\ldots \vdash s_{15}\#_2\#_1 O\#_0 bcc\ [s_{14} \triangledown s_{15}] \tag{2.30}$$

$$\ldots \vdash s_{13}\#_2\#_1 ccO\#_0 bcc\ [s_{15} \Upsilon\#_0 c s_{13}] \tag{2.31}$$

Automat vygeneruje neprázdný řetězec znaků b a porovná jej se vstupem:

$$\ldots \vdash s_{16}\#_1 bccO\#_0 bcc\ [s_{13} \oplus_1 b s_{16}] \tag{2.32}$$

A pokračuje v porovnávání až do konce:

$$\ldots \vdash s_{18}\#_1 ccO\#_0 cc\ [s_{16} \triangledown s_{18}] \tag{2.33}$$

$$\ldots \vdash s_{18}\#_1 cO\#_0 c\ [s_{18} \triangledown s_{18}] \tag{2.34}$$

$$\ldots \vdash s_{18}\#_1 O\#_0\ [s_{18} \triangledown s_{18}] \tag{2.35}$$

$$\ldots \vdash s_{19}\#_1 OO\#_0\ [s_{18} \Upsilon\#_0 O s_{19}] \tag{2.36}$$

2.5 Neformální popis

Omezený Hromádkův automat je rozšířením zásobníkového automatu. Obsahuje konečné stavové řízení spolu s hlavním zásobníkem, který má tytéž možnosti jako zásobník zásobníkového automatu (s pohodlnější detekcí dna zásobníku), a navíc umožňuje vytvářet další číselně označené zásobníky, vkládat do nich vstupní symboly, spojovat je a porovnávat vzniklý řetězec vstupních symbolů se vstupem.

2.5.1 Konfigurace

Konfigurace omezeného Hromádkova automatu je reprezentována jako řetězec symbolů. První znak tohoto řetězce náleží do množiny stavů Q a představuje aktuální stav automatu v dané konfiguraci. Následuje obsah všech zásobníků v pořadí od zásobníku s nejvyšším číselným označením po hlavní zásobník (s číselným označením 0).

Každý zásobník číslo i je ohraničen zleva symbolem $\#_{i+1}$ a zprava symbolem $\#_i$ a jeho obsah je zapsán v pořadí od vrcholu zásobníku po dno. Hlavní zásobník může obsahovat vedle znaků vstupní abecedy i neterminální symboly ze Z, ostatní zásobníky mohou obsahovat pouze znaky vstupní abecedy Σ.

Za znakem $\#_0$ následuje dosud nenačtená část vstupního řetězce v přirozeném pořadí symbolů. *Vnitřní konfigurací* se rozumí konfigurace bez vstupního řetězce.

2.5.2 Pravidla

Množina R může obsahovat pravidla sedmi tříd podle tabulky 2.1. U všech tříd pravidel platí, že $p_1, p_2 \in Q$ a automat při derivačním kroku podle daného pravidla přejde ze stavu p_1 do stavu p_2 (v jiném stavu než p_1 nelze dané pravidlo použít).

1. Pravidlo první třídy má tvar $p_1 c \#_0 u p_2$, kde $c \in (\Sigma \cup Z)$ a $u \in (\Sigma \cup Z)^*$. Jde o pravidlo přepisovací. Pokud se na vrcholu hlavního zásobníku nachází symbol c, bude při provedení derivačního kroku podle tohoto pravidla přepsán řetězcem u.

2. Pravidlo druhé třídy má tvar $p_1 \Upsilon \#_0 u p_2$, kde $u \in (\Sigma \cup Z)^*$. Při provedení derivačního kroku podle tohoto pravidla bude na vrchol hlavního zásobníku vložen řetězec u.

3. Pravidlo třetí třídy má tvar $p_1 \Omega \#_0 u p_2$ a stejný význam jako pravidlo druhé třídy, až na to, že může být použito, pouze pokud je hlavní zásobník prázdný.

4. Pravidlo čtvrté třídy má tvar $p_1 \#_i u p_2$, kde $0 < i < n$ a $u \in \Sigma^*$. Toto pravidlo umožňuje vkládání řetězců symbolů vstupní abecedy do jiných zásobníků než do hlavního. Na rozdíl od vkládání do hlavního zásobníku, použití tohoto pravidla nelze omezit obsahem zásobníku, do kterého vkládá, je omezeno pouze stavem automatu a aktuálním počtem zásobníků (pokud cílový zásobník neexistuje, nemůžeme do něj vkládat symboly).

5. Pravidlo páté třídy má tvar $p_1 \bigtriangledown p_2$ a jedná se o pravidlo porovnání. Při derivačním kroku podle tohoto pravidla automat porovná symbol na vrcholu zásobníku s nejvyšším číselným označením s prvním nenačteným symbolem vstupního řetězce, a pokud se shodují, symbol ze zásobníku vyjme a symbol vstupního řetězce načte (takže se bude příště porovnávat další symbol vstupního řetězce). Derivační krok podle pravidla páté třídy nemůže být proveden, pokud platí alespoň jedna z následujících podmínek:

- Na vrcholu zásobníku s nejvyšším číselným označením je vstupní symbol jiný než je první nenačtený symbol vstupního řetězce.
- Zásobník s nejvyšším číselným označením je prázdný (na obsahu jiných zásobníků nezáleží).
- V automatu je jen hlavní zásobník a na jeho vrcholu se nachází neterminál.
- Byl již načten celý vstupní řetězec.

6. Pravidlo šesté třídy má tvar $p_1 \otimes p_2$. Jde o jediné pravidlo, kterým vznikají nové zásobníky. Při provedení derivačního kroku podle tohoto pravidla se všechny zásobníky kromě hlavního přečíslují na číselné označení o 1 vyšší a vloží se nový zásobník číslo 1, do kterého se přemístí všechny terminální symboly z vrcholu hlavního zásobníku až po první neterminální symbol, nebo úplně všechny, pokud hlavní zásobník neterminální symbol neobsahuje. Například pokud má hlavní zásobník tvar $\#_1 abcMdNef\#_0$, kde malá písmena označují terminální symboly a velká písmena neterminály, bude mít nový zásobník číslo 1 tvar $\#_2 abc\#_1$ a hlavní zásobník bude mít po provedení derivačního kroku tvar $\#_1 MdNef\#_0$. Pravidlo šesté třídy nelze aplikovat, pokud má automat n zásobníků.

7. Pravidlo sedmé třídy má tvar $p_1 \oplus_i v p_2$, kde $0 < i < n$ a $v \in \Sigma.\Sigma^*$. Je o pravidlo spojení zásobníků, jediné pravidlo, kterým může počet zásobníků v automatu klesnout. Při derivačním kroku podle tohoto pravidla budou zásobníky číslo i a $i-1$ spojeny se současným vložením neprázdného řetězce vstupních symbolů v. Toto pravidlo nelze použít, pokud má automat nejvýše i zásobníků.

Kapitola 3

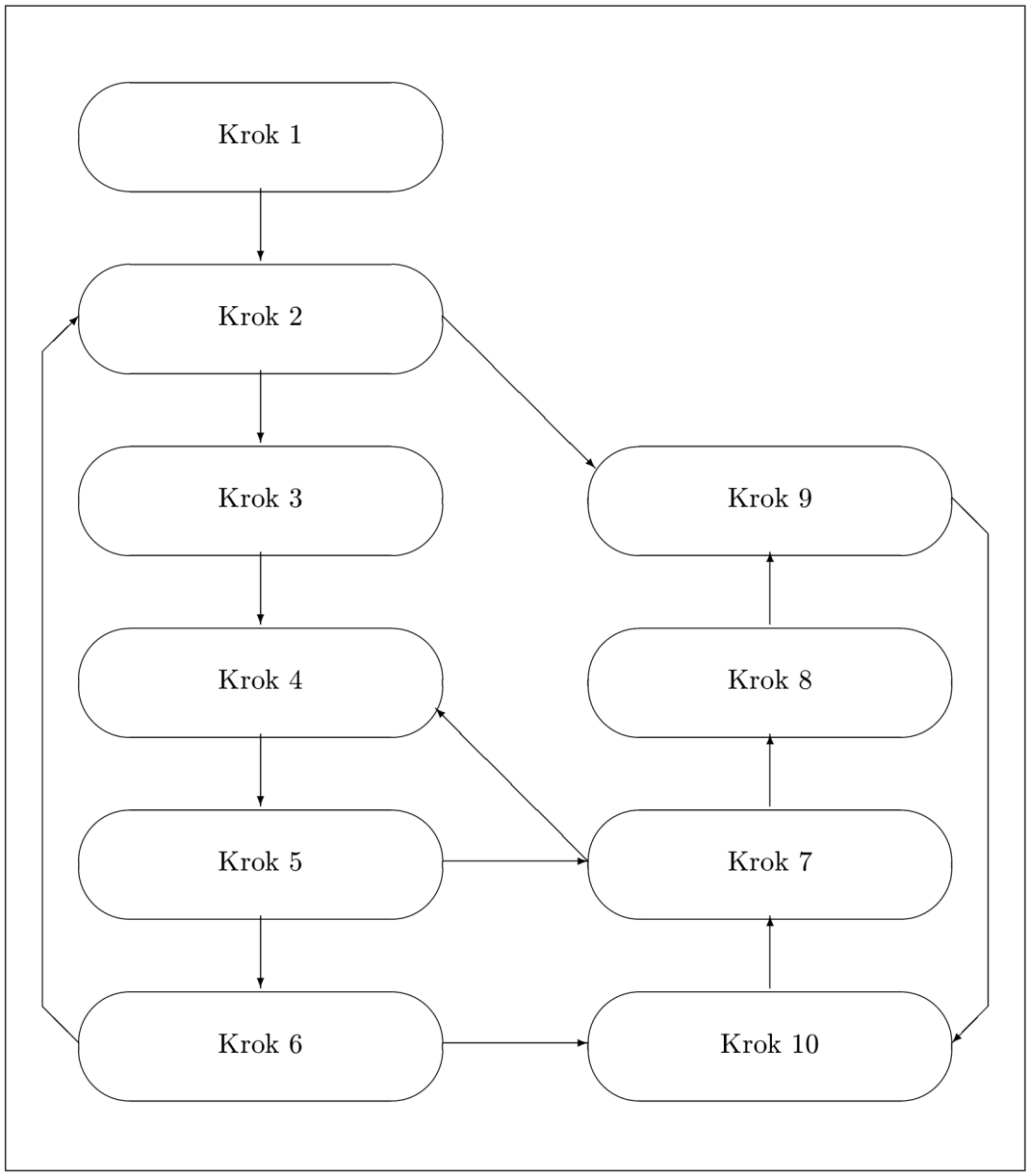
Použitý algoritmus

Úkolem implementované aplikace je zjistit, zda zadaný omezený Hromádkův automat přijme určitý vstupní řetězec, tedy zda existuje konečná posloupnost derivačních kroků, kterými automat přejde z počáteční vnitřní konfigurace k do některé koncové vnitřní konfigurace $f \in F$ za současného přečtení celého vstupního řetězce.

K dosažení tohoto cíle aplikace používá metodu slepého prohledávání stavového prostoru backtracking aplikovanou na prohledávání prostoru konfigurací automatu:

1. Nastav automat do počáteční vnitřní konfigurace (včetně stavu).
2. Vyber první pravidlo z uspořádané množiny pravidel vycházejících z aktuálního stavu automatu. Pokud takové pravidlo neexistuje (množina je prázdná), jdi na krok 9.
3. Vlož pravidlo vybrané v kroku 2 na vrchol zásobníku operací (dále jen *zásobník*).
4. Vezmi pravidlo, na které je odkaz na vrcholu zásobníku a pokus se jej aplikovat na aktuální konfiguraci automatu (dále jen *konfigurace*). V případě úspěchu jdi na krok 5, jinak jdi na krok 7.
5. Pokud již byl přečten celý vstupní řetězec a aktuální konfigurace se rovná jedné z koncových vnitřních konfigurací, vrať obsah zásobníku a skonči úspěchem.
6. Pokud není zásobník plný, jdi na krok 2, jinak jdi na krok 10.
7. Nahraď vrchol zásobníku odkazem na další pravidlo z uspořádané množiny pravidel vycházejících z aktuálního stavu. Pokud takové pravidlo existuje (toto nebylo poslední), jdi na krok 4, jinak jdi na krok 8.
8. Vyjmi vrchol zásobníku.
9. Pokud je zásobník prázdný, skonči neúspěchem.
10. Proveď s konfigurací inverzní operaci k pravidlu, na které je odkaz na vrcholu zásobníku.
11 Jdi na krok 7.

Pravidla automatu jsou při načítání roztříděna podle jejich výchozího stavu a uspořádána do uspořádaných množin, což programu umožňuje snížit počet pravidel, kterými se musí v jednotlivé fázi procesu prohledávání zabývat.



Obrázek 3.1: Schéma algoritmu

3.1 Komentáře k algoritmu

Ke kroku 1 Program bude simulovat derivační kroky automatu nad datovou reprezentací jeho vnitřní konfigurace a vstupního řetězce. Tato reprezentace je nastavena do počátečního stavu během načítání souboru se specifikací automatu.

Ke kroku 2 V tomto kroku se z příslušné uspořádané množiny vybere první z pravidel. Pokud aplikace tohoto pravidla nepovede k načtení celého vstupního řetězce a přechodu do nějaké koncové konfigurace, bude program později vybírat další pravidla.

Ke kroku 3 Zásobník operací je zásobník s pevně danou maximální velikostí, do kterého jsou ukládány jednotlivé operace programu. Operací se rozumí jednak odkaz na pravidlo, s kterým byl proveden příslušný derivační krok, a dále data potřebná k uvedení konfigurace do původního stavu (to znamená do stavu před provedením daného derivačního kroku).

Ke kroku 4 Na vrcholu zásobníku operací je odkaz na nějaké pravidlo. Aplikace se pokusí provést v automatu derivační krok s použitím tohoto pravidla. Pokud toto nebude možné, přejde aplikace k dalšímu pravidlu. Pokud se aplikace pravidla podaří, automat tím přejde do nové konfigurace.

Ke kroku 5 Po provedení každého derivačního kroku automat kontroluje, zda byl přečten celý vstupní řetězec, a pokud ano, také kontroluje, zda nebyla dosažená některá koncová konfigurace. Pokud jsou obě tyto podmínky splněny, na zásobníku operací se nachází posloupnost pravidel vedoucí k přijetí vstupního řetězce, a tato posloupnost bude spolu s prohlášením o přijetí vstupního řetězce výstupem programu.

Ke kroku 6 Zásobník operací má limitovanou velikost. Jedná se o nezbytnou ochranu proti zacyklení. Pokud je zásobník plný, nebude se již automat pokoušet aplikovat další pravidlo. V důsledku toho nenajde přijímající posloupnosti derivačních kroků delší, než jaké je omezení velikosti zásobníku operací. Toto omezení je však nezbytné. Tento limit je stanoven při spuštění programu.

Ke kroku 7 Nyní platí, že pokus o aplikaci předchozího pravidla byl neúspěšný, nebo sice byl úspěšný, ale nevedl k nalezení přijímající posloupnosti derivačních kroků. V obou případech aplikace pokračuje v prohledávání stavového prostoru aplikací dalšího pravidla. Pokud ovšem další aplikovatelné pravidlo neexistuje, provede aplikace zpětný krok.

Ke kroku 8 Na vrcholu zásobníku je poslední pravidlo, jehož výchozím stavem je aktuální stav automatu. Aplikace provádí zpětný krok, takže toto pravidlo musí být vyjmuto.

Ke kroku 9 Pokud dospěl program při zpětném kroku k vyprázdnění zásobníku operací, je hledání neúspěšné.

Ke kroku 10 V tomto kroku se provádí samotný zpětný krok – automat se tedy vrátí do konfigurace před aplikací určitého pravidla, protože žádná prohledaná posloupnost derivačních kroků následující po aplikacei tohoto pravidla nevedla k přijetí vstupního řetězce.

Ke kroku 11 Postup k dalšímu pravidlu v rámci daného stavu.

Pokud existuje více posloupností derivačních kroků, kterými omezený Hromádkův automat přijme zkoumaný vstupní řetězec, uvedený algoritmus najde a vrátí pouze jednu z nich (kterou, to závisí na uspořádání pravidel v uspořádaných množinách).

Drobnou úpravou algoritmu by bylo možné zajistit úplné prohledání stavového podprostoru daného limitovanou velikostí zásobníku operací a nalezení všech takových posloupností s určitou maximální délkou.

3.2 Proč je nutno velikost zásobníku operací limitovat?

V automatu se mohou vyskytnout pravidla jako např. $s_1\#_1 a s_1$ nebo $s_1 \Upsilon \#_0 s_1$, která je možno aplikovat opakovaně bez nějakého limitu. Aby se program nezacyklil, je nutno omezit maximální velikost zásobníku operací. Toto omezení je obdobné jako u algoritmu prohledávání do omezené hloubky DLS.

3.3 Možné cesty dalšího vývoje

3.3.1 Upřednostnění porovnání

Upřednostněním pravidel porovnání v uspořádaných množinách lze dosáhnout rychlejšího nalezení přijímající posloupnosti, protože program bude nejprve zkoumat vstup a teprve poté vkládat symboly na zásobníky.

3.3.2 Nalezení nejkratší posloupnosti

Při nalezení přijímající posloupnosti by bylo možno tuto uložit a ve vyhledávání pokračovat s tím, že by bylo stanoveno nové omezení velikosti zásobníku operací na velikost o jeden krok menší než je nalezená posloupnost. V důsledku toho by pak program nebyl schopen najít žádnou další alespoň stejně dlouhou přijímající posloupnost derivačních kroků, ale byl by schopen najít kratší posloupnosti. Po dokončení prohledávacího algoritmu by pak bylo možno tvrdit, že poslední nalezená přijímající posloupnost je globálně nejkratší.

Kapitola 4

Příklady

Mějme jazyk $L(A) = \{a^n b^n c^n \mid n \geq 1\}$. Tento jazyk je přijímán omezeným Hromádkovým automatem $A = (Q, \Sigma, Z, R, k, F, n)$, kde:

$$
\begin{aligned}
Q &= \{s_1, s_2, s_3, s_4, s_5, s_6, s_7\} \\
\Sigma &= \{a, b, c\} \\
Z &= \{z \textit{ (zásobníková abeceda je nevyužita)}\} \\
R &= \{s_1 \otimes s_2, s_2 \otimes s_3, s_3 \#_2 a s_4, s_4 \oplus_2 b s_5, s_5 \oplus_1 c s_6, s_6 \bigtriangledown s_6, s_4 \#_1 b s_7, s_7 \Upsilon \#_0 c s_3\} \\
k &= s_1 \#_1 \#_0 \\
F &= \{s_6 \#_1 \#_0\} \\
n &= 3
\end{aligned}
$$

Zde je příklad, jak tento automat přijme řetězec *aaabbbccc*:

Konfigurace	Pravidlo
$s_1\#_1\#_0aaabbbccc$	$s_1 \otimes s_2$
$s_2\#_2\#_1\#_0aaabbbccc$	$s_2 \otimes s_3$
$s_3\#_3\#_2\#_1\#_0aaabbbccc$	$s_3\#_2as_4$
$s_4\#_3a\#_2\#_1\#_0aaabbbccc$	$s_4\#_1bs_7$
$s_7\#_3a\#_2b\#_1\#_0aaabbbccc$	$s_7\Upsilon\#_0cs_3$
$s_3\#_3a\#_2b\#_1c\#_0aaabbbccc$	$s_3\#_2as_4$
$s_4\#_3aa\#_2b\#_1c\#_0aaabbbccc$	$s_4\#_1bs_7$
$s_7\#_3aa\#_2bb\#_1c\#_0aaabbbccc$	$s_7\Upsilon\#_0cs_3$
$s_3\#_3aa\#_2bb\#_1cc\#_0aaabbbccc$	$s_3\#_2as_4$
$s_4\#_3aaa\#_2bb\#_1cc\#_0aaabbbccc$	$s_4 \oplus_2 bs_5$
$s_5\#_2aaabbb\#_1cc\#_0aaabbbccc$	$s_5 \oplus_1 cs_6$
$s_6\#_1aaabbbccc\#_0aaabbbccc$	$s_6 \bigtriangledown s_6$
$s_6\#_1aabbbccc\#_0aabbbccc$	$s_6 \bigtriangledown s_6$
$s_6\#_1abbbccc\#_0abbbccc$	$s_6 \bigtriangledown s_6$
$s_6\#_1bbbccc\#_0bbbccc$	$s_6 \bigtriangledown s_6$
$s_6\#_1bbccc\#_0bbccc$	$s_6 \bigtriangledown s_6$
$s_6\#_1bccc\#_0bccc$	$s_6 \bigtriangledown s_6$
$s_6\#_1ccc\#_0ccc$	$s_6 \bigtriangledown s_6$
$s_6\#_1cc\#_0cc$	$s_6 \bigtriangledown s_6$
$s_6\#_1c\#_0c$	$s_6 \bigtriangledown s_6$
$s_6\#_1\#_0$	

Mějme jazyk $L(A) = \{(a^o b^o)^p \mid o, p \in \mathbb{N}\}$. Tento jazyk je přijímán omezeným Hromádkovým automatem $A = (Q, \Sigma, Z, R, k, F, n)$, kde:

$$
\begin{aligned}
Q &= \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}, s_{12}, s_{13}\} \\
\Sigma &= \{a, b\} \\
Z &= \{M\} \\
R &= \{s_1 \Upsilon \#_0 M s_1, s_1 M \#_0 s_2, s_2 M \#_0 s_3, s_2 \Omega \#_0 s_5, s_3 \#_2 a s_4, s_4 \#_1 b s_2, s_5 \oplus_2 ab s_6, \\
&\quad s_6 \bigtriangledown s_7, s_7 \bigtriangledown s_8, s_8 \Upsilon \#_0 M s_6, s_8 \oplus_1 a s_9, s_9 a \#_0 M s_{10}, s_{10} \otimes s_{11}, s_{10} M \#_0 s_{13}, \\
&\quad s_{11} \otimes s_{12}, s_{12} M \#_0 s_2, s_{13} M \#_0 s_{13}\} \\
k &= s_1 \#_3 \#_2 \#_1 \#_0 \\
F &= \{s_{13} \#_1 \#_0\} \\
n &= 3
\end{aligned}
$$

Zde je příklad, jak tento automat přijme řetězec ab:

$$s_1 \#_3 \#_2 \#_1 \#_0 ab \vdash s_1 \#_3 \#_2 \#_1 M \#_0 ab \; [s_1 M \#_0 s_2] \tag{4.1}$$

$$\ldots \vdash s_2 \#_3 \#_2 \#_1 \#_0 ab \; [s_1 M \#_0 s_2] \tag{4.2}$$

$$\ldots \vdash s_5 \#_3 \#_2 \#_1 \#_0 ab \; [s_2 \Omega \#_0 s_5] \tag{4.3}$$

$$\ldots \vdash s_6 \#_2 ab \#_1 \#_0 ab \; [s_5 \oplus_2 ab s_6] \tag{4.4}$$

$$\ldots \vdash s_7 \#_2 b \#_1 \#_0 b \; [s_6 \bigtriangledown s_7] \tag{4.5}$$

$$\ldots \vdash s_8 \#_2 \#_1 \#_0 \; [s_7 \bigtriangledown s_8] \tag{4.6}$$

$$\ldots \vdash s_9 \#_1 a \#_0 \; [s_8 \oplus_1 a s_9] \tag{4.7}$$

$$\ldots \vdash s_{10} \#_1 M \#_0 \; [s_9 a \#_0 M s_{10}] \tag{4.8}$$

$$\ldots \vdash s_{13} \#_1 \#_0 \; [s_{10} M \#_0 s_{13}] \tag{4.9}$$

A jak přijme řetězec $aabbaabb$:

Automat nejprve ve stavu s_1 vygeneruje na hlavním zásobníku o symbolů M a přejde do stavu s_2.

$$s_1 \#_3 \#_2 \#_1 \#_0 ab \vdash s_1 \#_3 \#_2 \#_1 M \#_0 aabbaabb \; [s_1 \Upsilon \#_0 M s_1] \tag{4.10}$$

$$\ldots \vdash s_1 \#_3 \#_2 \#_1 M M \#_0 aabbaabb \; [s_1 \Upsilon \#_0 M s_1] \tag{4.11}$$

$$\ldots \vdash s_2 \#_3 \#_2 \#_1 M \#_0 aabbaabb \; [s_1 M \#_0 s_2] \tag{4.12}$$

S použitím zásobníků číslo 1 a 2 automat vygeneruje řetězec $a^o b^o$.

$$\ldots \vdash s_3 \#_3 \#_2 \#_1 \#_0 aabbaabb \; [s_2 M \#_0 s_3] \tag{4.13}$$

$$\ldots \vdash s_4 \#_3 a \#_2 \#_1 \#_0 aabbaabb \; [s_3 \#_2 a s_4] \tag{4.14}$$

$$\ldots \vdash s_2 \#_3 a \#_2 b \#_1 \#_0 aabbaabb \; [s_4 \#_1 b s_2] \tag{4.15}$$

$$\ldots \vdash s_5 \#_3 a \#_2 b \#_1 \#_0 aabbaabb \; [s_2 \Omega \#_0 s_5] \tag{4.16}$$

$$\ldots \vdash s_6 \#_2 aabb \#_1 \#_0 aabbaabb \; [s_5 \oplus_2 ab s_6] \tag{4.17}$$

Vygenerovaný řetězec nyní porovná se vstupem a přitom zároveň vrátí na hlavní zásobník o symbolů M.

$$\ldots \vdash s_7\#_2abb\#_1\#_0abbaabb\ [s_6 \bigtriangledown s_7] \tag{4.18}$$

$$\ldots \vdash s_8\#_2bb\#_1\#_0bbaabb\ [s_7 \bigtriangledown s_8] \tag{4.19}$$

$$\ldots \vdash s_6\#_2bb\#_1M\#_0bbaabb\ [s_8 \Upsilon \#_0 M s_6] \tag{4.20}$$

$$\ldots \vdash s_7\#_2b\#_1M\#_0baabb\ [s_6 \bigtriangledown s_7] \tag{4.21}$$

$$\ldots \vdash s_8\#_2\#_1M\#_0aabb\ [s_7 \bigtriangledown s_8] \tag{4.22}$$

$$\ldots \vdash s_9\#_1aM\#_0aabb\ [s_8 \oplus_1 as_9] \tag{4.23}$$

$$\ldots \vdash s_{10}\#_1MM\#_0aabb\ [s_9a\#_0Ms_{10}] \tag{4.24}$$

$$\ldots \vdash s_{11}\#_2\#_1MM\#_0aabb\ [s_{10} \otimes s_{11}] \tag{4.25}$$

$$\ldots \vdash s_{12}\#_3\#_2\#_1MM\#_0aabb\ [s_{11} \otimes s_{12}] \tag{4.26}$$

Automat se vrátí do stavu s_2 a svůj postup zopakuje ještě $(o-1)$krát.

$$\ldots \vdash s_2\#_3\#_2\#_1M\#_0aabb\ [s_{12}M\#_0s_2] \tag{4.27}$$

$$\ldots \vdash s_3\#_3\#_2\#_1\#_0aabb\ [s_2M\#_0s_3] \tag{4.28}$$

$$\ldots \vdash s_4\#_3a\#_2\#_1\#_0aabb\ [s_3\#_2as_4] \tag{4.29}$$

$$\ldots \vdash s_2\#_3a\#_2b\#_1\#_0aabb\ [s_4\#_1bs_2] \tag{4.30}$$

$$\ldots \vdash s_5\#_3a\#_2b\#_1\#_0aabb\ [s_2\Omega\#_0s_5] \tag{4.31}$$

$$\ldots \vdash s_6\#_2aabb\#_1\#_0aabb\ [s_5 \oplus_2 abs_6] \tag{4.32}$$

$$\ldots \vdash s_7\#_2abb\#_1\#_0abb\ [s_6 \bigtriangledown s_7] \tag{4.33}$$

$$\ldots \vdash s_8\#_2bb\#_1\#_0bb\ [s_7 \bigtriangledown s_8] \tag{4.34}$$

$$\ldots \vdash s_6\#_2bb\#_1M\#_0bb\ [s_8 \Upsilon \#_0 M s_6] \tag{4.35}$$

$$\ldots \vdash s_7\#_2b\#_1M\#_0b\ [s_6 \bigtriangledown s_7] \tag{4.36}$$

$$\ldots \vdash s_8\#_2\#_1M\#_0\ [s_7 \bigtriangledown s_8] \tag{4.37}$$

$$\ldots \vdash s_9\#_1aM\#_0\ [s_8 \oplus_1 as_9] \tag{4.38}$$

$$\ldots \vdash s_{10}\#_1MM\#_0\ [s_9a\#_0Ms_{10}] \tag{4.39}$$

Automat přejde do stavu s_{13}, v němž pohltí všechna zbylá M, čímž přejde do koncové konfigurace.

$$\ldots \vdash s_{13}\#_1M\#_0\ [s_{10}M\#_0s_{13}] \tag{4.40}$$

$$\ldots \vdash s_{13}\#_1\#_0\ [s_{13}M\#_0s_{13}] \tag{4.41}$$

Kapitola 5

Implementace

Pro účely programu jsou omezený Hromádkův automat a jeho vstupní řetězec reprezentovány textovým souborem přesně daného formátu. Načtení tohoto souboru je nezbytné před spuštěním samotného algoritmu prohledávání stavového prostoru a je předmětem lexikální, syntaktické a sémantické analýzy.

5.1 Postup implementace

Návrh aplikace byl proveden metodou shora dolů. Nejprve byly navrženy datové struktury pro reprezentaci omezeného Hromádkova automatu, zásobníku operací a lexikální analýzu. Byl navržen formát vstupního souboru a popsán pomocí konečných stavových automatů.

V následujícím kroku bylo přistoupeno k implementaci. Implementace byla provedena v programovacím jazyce C++. Jako první byla implementována lexikální analýza, jejímž jádrem je funkce `get_token(struct token *, FILE *)`, která načítá znaky ze vstupního souboru a sestavuje z nich tokeny pro syntaktickou analýzu.

Pak byla implementována syntaktická a sémantická analýza, která je soustředěna v modulu `au.cc`, zejména pak ve funkci `load(oha *, fsm_input *, FILE *)`, jejímž úkolem je načíst datovou reprezentaci omezeného Hromádkova automatu ze vstupního souboru. Sémantická analýza se zde prolíná se syntaktickou a je představována například kontrolami, zda znaky v zásobnících náleží do vstupní abecedy, zda počet zásobníků v některé uvedené konfiguraci nepřesahuje n a podobně. Díky vhodnému návrhu formátu vstupního souboru je možno tyto kontroly provádět okamžitě při načítání a není nutno provádět další kontroly po načtení celého vstupního souboru.

Dále byl implementován zásobník operací, který je reprezentován abstraktním datovým typem `opst`, který poskytuje operace obvyklé u abstraktního datového typu zásobník a dále operaci *full*(), která poskytuje informaci, zda již byl dosažen maximální počet prvků.

Algoritmus prohledávání stavového prostoru, který tvoří funkční jádro aplikace, byl implementován ve funkci `hledej(oha *, fsm_input *, opst *)`, čímž byla implementační fáze projektu završena.

Další vývojovou fází bylo testování a ladění. Podle specifikace formátu vstupního souboru byl vypracován soubor reprezentující omezený Hromádkův automat přijímající jazyk $L(A) = \{a^n b^n c^n \mid n \geq 1\}$ a s tímto vstupním souborem byla aplikace testována.

Testování bylo úspěšné, neboť odhalilo řadu chyb, které bránily správné funkci aplikace a musely být opraveny.

5.2 Lexikální analýza

Lexikální analýza byla implementována pomocí rozšířeného deterministického konečného stavového automatu. Lexikální analyzátor rozeznává tokeny typů uvedených v tabulce 5.1 (regulární výraz `\A` představuje znak nového řádku). Konečný stavový automat A lexikál-

Typ tokenu	Symbolicky	Rozšířený regulární výraz
Číslo	(n)	`\([0-9]+\)`
Terminál	a	`[0-9A-Za-z]|\\x[0-9A-Fa-f]{4}`
Neterminál	N	`<([0-9A-Za-z]|\\x[0-9A-Fa-f]{4})>`
Mříž	#	`#`
Kl. slovo always	\"always"	`\\"always"`
Kl. slovo empty	\"empty"	`\\"empty"`
Kl. slovo join	\"join"	`\\"join"`
Kl. slovo cmp	\"cmp"	`\\"cmp"`
Kl. slovo cut	\"cut"	`\\"cut"`
Konec řádku	↩	`\A`

Tabulka 5.1: Typy tokenů

ního analyzátoru je definován následovně:

$$A = (Q, \Sigma, (\text{ST_BEGIN}), R, F)$$

Kde:

- Množina stavů $Q = \{$(ST_BEGIN), (ST_NUMBER), (ST_NONTERM), (ST_IMMNT), (ST_HEXNT), (ST_BACKSLASH), (ST_KW), (ST_KW_A), (ST_KW_C), (ST_KW_E), (ST_KW_J), (EST_LF), (EST_HASH), (EST_IMMCHAR), (EST_NUMBER), (EST_IMMNT), (EST_HEXNT), (EST_HEXCHAR), (EST_KW_ALWAYS), (EST_KW_EMPTY), (EST_KW_JOIN), (EST_KW_CMP), (EST_KW_CUT)$\}$.
- Vstupní abeceda $\Sigma = \{\hookleftarrow, \text{␣}, \text{”}, \#, (,), 0, 1, \ldots, 9, <, >, \text{A}, \text{B}, \ldots, \text{Z}, \backslash, \text{a}, \text{b}, \ldots, \text{z}\}$.
- (ST_BEGIN) je počáteční stav.
- Množina pravidel R je uvedena v tabulce 5.2.
- Množina koncových stavů $F = \{$(EST_LF), (EST_HASH), (EST_IMMCHAR), (EST_NUMBER), (EST_IMMNT), (EST_HEXNT), (EST_HEXCHAR), (EST_KW_ALWAYS), (EST_KW_EMPTY), (EST_KW_JOIN), (EST_KW_CMP), (EST_KW_CUT)$\}$.

Pro přehlednost mají nekoncové stavy předponu ST_ a koncové stavy předponu EST_.

{(ST_BEGIN)	↩	⟶	(EST_LF)}
{(ST_BEGIN)	\	⟶	(ST_BACKSLASH)}
{(ST_BEGIN)	<	⟶	(ST_NONTERM)}
{(ST_BEGIN)	(	⟶	(ST_NUMBER)}
{(ST_BEGIN)	␣	⟶	(ST_BEGIN)}
{(ST_BEGIN)	[0-9A-Za-z]	⟶	(EST_IMMCHAR)}
{(ST_NUMBER)	[0-9]	⟶	(ST_NUMBER)}
{(ST_NUMBER)	)	⟶	(EST_NUMBER)}
{(ST_NONTERM)	\	⟶	(ST_HEXNT)}
{(ST_NONTERM)	[0-9A-Za-z]	⟶	(ST_IMMNT)}
{(ST_IMMNT)	>	⟶	(EST_IMMNT)}
{(ST_HEXNT)	[0-9A-Fa-f]{4}>	⟶	(EST_HEXNT)}
{(ST_BACKSLASH)	”	⟶	(ST_KW)}
{(ST_BACKSLASH)	x[0-9A-Fa-f]{4}	⟶	(EST_HEXCHAR)}
{(ST_BACKSLASH)	↩	⟶	(ST_BEGIN)}
{(ST_KW)	a	⟶	(ST_KW_A)}
{(ST_KW)	e	⟶	(ST_KW_E)}
{(ST_KW)	c	⟶	(ST_KW_C)}
{(ST_KW)	j	⟶	(ST_KW_J)}
{(ST_KW_A)	lways	⟶	(EST_KW_ALWAYS)}
{(ST_KW_E)	mpty	⟶	(EST_KW_EMPTY)}
{(ST_KW_C)	mp	⟶	(EST_KW_CMP)}
{(ST_KW_C)	ut	⟶	(EST_KW_CUT)}
{(ST_KW_J)	oin	⟶	(EST_KW_JOIN)}

Tabulka 5.2: Množina pravidel lexikální analýzy

5.3 Syntaktická a sémantická analýza

Syntaxe vstupního souboru je popsána bezkontextovou gramatikou: $G = (N, T, <\text{program}>, R)$, kde:

- Množina neterminálů $N = \{$<hlavní zásobník>, <jedna konfigurace>, <neterminály>, <podmínka>, <pravidla>, <pravidlo>, <program>, <seznam konfigurací>, <vstupní řetězec>, <zásobníky>, <znaky vstupní abecedy>$\}$.
- Množina terminálů T je tvořena typy tokenů podle tabulky 5.1.
- <program> je počáteční neterminál.
- Množina přepisovacích pravidel R je uvedena v tabulce 5.3.

<hlavní zásobník>	$\longrightarrow$	ε
<hlavní zásobník>	$\longrightarrow$	a<hlavní zásobník>
<hlavní zásobník>	$\longrightarrow$	N<hlavní zásobník>
<jedna konfigurace>	$\longrightarrow$	(n)<zásobníky>#(1)<hlavní zásobník>#
<neterminály>	$\longrightarrow$	ε
<neterminály>	$\longrightarrow$	N<neterminály>
<podmínka>	$\longrightarrow$	a
<podmínka>	$\longrightarrow$	N
<podmínka>	$\longrightarrow$	\"empty"
<podmínka>	$\longrightarrow$	\"always"
<pravidla>	$\longrightarrow$	ε
<pravidla>	$\longrightarrow$	<pravidlo>↩<pravidla>
<pravidlo>	$\longrightarrow$	(n)<podmínka>#(0)<hlavní zásobník>(n)
<pravidlo>	$\longrightarrow$	(n)#(n)<znaky vstupní abecedy>(n)
<pravidlo>	$\longrightarrow$	(n)\"cmp"(n)
<pravidlo>	$\longrightarrow$	(n)\"cut"(n)
<pravidlo>	$\longrightarrow$	(n)\"join"(n)a<znaky vstupní abecedy>(n)
<program>	$\longrightarrow$	(n)(n)a<znaky vstupní abecedy>↩ N<neterminály>↩<pravidla>↩ <seznam konfigurací>↩<jedna konfigurace>↩ <vstupní řetězec>#
<seznam konfigurací>	$\longrightarrow$	ε
<seznam konfigurací>	$\longrightarrow$	<jedna konfigurace>↩<seznam konfigurací>
<vstupní řetězec>	$\longrightarrow$	ε
<vstupní řetězec>	$\longrightarrow$	a<vstupní řetězec>
<vstupní řetězec>	$\longrightarrow$	↩<vstupní řetězec>
<zásobníky>	$\longrightarrow$	ε
<zásobníky>	$\longrightarrow$	#<znaky vstupní abecedy><zásobníky>
<znaky vstupní abecedy>	$\longrightarrow$	ε
<znaky vstupní abecedy>	$\longrightarrow$	a<znaky vstupní abecedy>

Tabulka 5.3: Pravidla gramatiky

Formát vstupního souboru byl navržen tak, aby jej bylo možno načíst a zpracovat pomocí obyčejného konečného stavového automatu, což významně usnadnilo implementaci. V im-

plementaci je použit podautomat, ale pouze proto, aby nemusela být opakována poměrně velká část kódu načítající vnitřní konfiguraci automatu.

Sémantická analýza je realizována při načítání a jejím úkolem je především odhalit sémantické chyby ve vstupním souboru. Zejména se kontroluje:

- Zda zadané $n > 0$.
- Zda není zadáno pravidlo ve tvaru $p_1 \#_0 u p_2$.
- Zda všechna zadaná čísla zásobníků (v pravidlech i konfiguracích) jsou menší než n.
- Zda všechna zadaná čísla stavů náleží do množiny stavů Q.
- Zda všechny zadané terminální symboly (v konfiguracích, ve vstupním řetězci) náleží do vstupní abecedy Σ.
- Zda všechny zadané neterminální symboly náleží do zásobníkové abecedy Z.

Opakování symbolů v definici abecedy terminálů či neterminálů není chybou, ale generuje varování vypsané na standardní chybový výstup. Nedostupné stavy v automatu nejsou chybou a nejsou kontrolovány.

Kapitola 6

Závěr

Tato bakalářská práce obohatila teorii formálních jazyků o *omezený Hromádkův automat*.

Před zahájením vlastní práce jsem nastudoval vybrané kapitoly z [6], kde jsem se seznámil s různými variantami zásobníkových automatů, gramatik a souvisejících nástrojů. Po důkladném obeznámení se s problematikou jsem vypracoval první verzi definice omezeného Hromádkova automatu. Tato verze počítala s možností číst symboly z čistých zásobníků. Podrobný průzkum však ukázal, že tato možnost nežádoucím způsobem zvyšuje výpočetní sílu automatu, a proto byla z definice vypuštěna a v definitivní definici je možné čtení pouze z hlavního zásobníku.

Dalším krokem byla analýza vlastností definovaného automatu. Při té příležitosti jsem vypracoval několik příkladů, které jsou náplní kapitoly 4. Ukázalo se, že konstrukce (sestavování) omezeného Hromádkova automatu produkuje obvykle poměrně značné množství stavů a pravidel. Na druhé straně poskytuje některé zajímavé možnosti, například počítání úspěšných porovnání s následným využitím získaného počtu ke konstrukci dalších řetězců. To umožnuje již při $n = 3$ přijímat jazyk $(a^o b^o)^p$, kde $o \in \mathbb{N}$ pro libovolné přirozené p. Hledáním jazyka, který by tímto automatem nebylo možné přijímat, jsem se nezabýval.

Dále jsem přešel z teoretické fáze do fáze praktické. Aplikoval jsem nově zavedený automat v oblasti syntaktické analýzy. Tato aplikace spočívala v návrhu aplikace činnost omezeného Hromádkova automatu simulující. Aplikace byla navržena tak, aby mohla simulovat jakýkoliv omezený Hromádkův automat s omezením na 65535 stavů, 65535 symbolů vstupní abecedy, 65535 neterminálních symbolů, 65536 stavů a maximální hloubku prohledávání 65535. Vzhledem k výpočetní náročnosti a použitému algoritmu je však i taková hloubka přehnaná a aplikace by měla rozumně pracovat při hloubce prohledávání nastavené nejvýše na 256.

Když byl návrh aplikace dostatečně propracovaný, přešel jsem do implementační fáze a zahájil jsem její implementaci v programovacím jazyce C++. Při této implementaci bylo s výhodou využito knihovny STL, která je součástí standardní knihovny jazyka C++[2].

Jak je popsáno v kapitole 5, omezený Hromádkův automat, jehož derivační kroky aplikace simuluje, je spolu se zadaným vstupním řetězcem reprezentován textovým souborem v přesně daném formátu, načítaným ze standardního vstupu. Proto je součástí aplikace lexikální a syntaktický analyzátor. Oba byly implementovány jako konečné stavové automaty. Lexikální analýza byla kompletně implementována a uzavřena již v první fázi implementace a později nebylo nutno do ní zasahovat. Implementace pokračovala syntaktickou analýzou s vestavěnými sémantickými kontrolami. Podrobnosti jsou uvedeny v kapitole 5. Po dokončení implementace načítání vstupního souboru, byl implementován zásobník operací a algoritmus umělé inteligence popsaný v kapitole 3, který představuje funkční jádro

celé aplikace. Pro účely ladění bylo nutno doplnit funkce schopné vypsat některé datové struktury na výstup.

Dalším krokem bylo testování implementované aplikace. Provedené testování bylo úspěšné, protože odhalilo několik závažných chyb v kódu, které bránily správné funkci aplikace. Tyto chyby byly opraveny a jejich opravy odladěny, aby aplikace správně zpracovala vhodně zvolený vzorový příklad.

6.1 O reprezentaci automatu

Není překvapivé, že konfigurace automatu je reprezentována řetězcem nad vhodnou abecedou. Překvapivé však může být, že jsou jako řetězce reprezentována i přechodová pravidla. Tato reprezentace je výhodná díky tomu, že nad řetězci existují některé užitečné operace jako například konkatenace. S řetězcem lze rovněž nakládat jako se seznamem symbolů. Řetězec je buď prázdný, nebo je tvořen svým prvním symbolem a zbytkem řetězce, který je rovněž řetězcem.

6.2 Zhodnocení dosažených výsledků

Z teoretického hlediska se navržený automat jeví být vhodným nástrojem pro popis některých formálních jazyků, na které bezkontextová gramatika nestačí. Sestavování a ověřování omezeného Hromádkova automatu pro určitý jazyk je pak dobré myšlenkové cvičení, které lze doporučit každému, kdo si potřebuje trénovat algoritmické uvažování.

Pro praktické využití má však automat několik nevýhod. Jednak nelze odstranit nedeterminismus, ten je bohužel ve stávající podobě automatu nevyhnutelný díky principu funkce operace porovnání. Nedeterminismus komplikuje implementaci a snižuje výkon výsledné aplikace. Druhým nedostatkem je velké množství kroků, stavů a pravidel, které musí omezený Hromádkův automat přijímající nějaký netriviální nebezkontextový jazyk mít.

Omezený Hromádkův automat je tedy především teoretický model s vyjadřovací silou vyšší než běžný zásobníkový automat. Jeho praktickému využití ale pravděpodobně brání výše uvedené nedostatky. Pokud by tyto nedostatky byly odstraněny, nebo alespoň zmírněny, dalo by se uvažovat o jeho nasazení i v praxi.

6.3 Možný budoucí vývoj projektu

V podstatě existují čtyři hlavní cesty, kterými se může vývoj projektu vydat.

První možností je přepracování definice automatu dodáním složitějších operací, které zvýší výpočetní sílu deterministické verze automatu tak, aby byla prakticky použitelná.

Druhou cestou je vylepšení implementující aplikace. Nabízí se možnost použití jiných metod umělé inteligence namísto backtrackingu, například A*, a vyvinutí sofistikovaných heuristických funkcí, jejichž použitím bude možno významně urychlit (a tím pádem optimalizovat) nalezení posloupnosti derivačních kroků, kterými omezený Hromádkův automat přijme zadaný vstupní řetězec.

Třetí možnost je kombinací první a druhé, čtvrtá cesta je cesta zapomnění – projekt zapadne a bude zapomenut.

Seznam použitých značek

Symbol	Strana	Popis
$A = B$	5	A se rovná B
(a, b, c)	5	uspořádaná trojice a, b, c
$A \subseteq B$	5	A je podmnožinou B
$A \in B$	5	A náleží do (je prvkem) B
$A \geq B$	5	A je větší nebo rovno B
$A \cup B$	5	sjednocení množin A a B
$A \leq B$	5	A je menší nebo rovno B
$\|A\|$	5	kardinalita množiny A
$A + B$	5	součet A a B
$l : A \to B$	5	l je zobrazení z A do B
Σ^*	5	iterace nad abecedou Σ
$A.B$	5	konkatenace řetězců A a B
2^A	5	potenční množina nad množinou A
$A - B$	5	aritmetický rozdíl A a B
$\bigcup_{i=1}^{n} f(i)$	5	sjednocení všech $f(i)$ pro $i = 1$, $i = 2, \ldots,$ $i = n$ včetně
ε	6	prázdný řetězec
$\longrightarrow$	20	přejít na/přepsat na (v pravidlech)

Literatura

[1] Chytil, M.: *Automaty a gramatiky.* SNTL, 1984, ISBN 3-540-61486-9.

[2] Josuttis, N. M.: *C++ Standardní knihovna a STL: Kompletní průvodce.* CP Books, 2005, ISBN 80-251-0511-1.

[3] Meduna, A.: *Automata and Languages : Theory and Applications.* Springer, 2000, ISBN 1-85233-074-0.

[4] Meduna, A.: Deep pushdown automata. *Acta Informatica*, ročník 42, č. 8-9, 2006: s. 541–552.

[5] Meduna, A.; Lukáš, R.: Formální jazyky a překladače (slidy k předmětu). 2011, přístupné jen studentům a zaměstnancům FIT VUT v Brně.

[6] Rozenberg, G.; Salomaa, A.: *Handbook of Formal Languages I.-III.* Springer-Verlag, 1997, ISBN 3-540-61486-9.

[7] *Zásobníkový automat – Wikipedie, otevřená encyklopedie* [online]. 12. 4. 2011 [cit. 2011-04-13], dostupný z WWW: <http://cs.wikipedia.org/wiki/Zásobníkový%20automat>.