VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

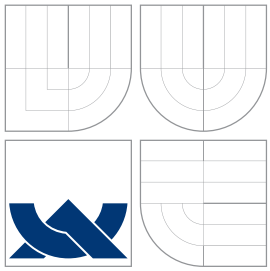
HLUBOKÉ ZÁSOBNÍKOVÉ AUTOMATY KONEČNÉHO INDEXU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

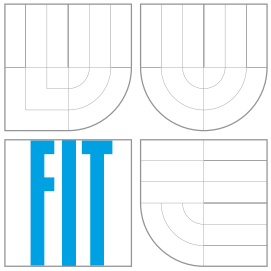
AUTOR PRÁCE
AUTHOR

VENDULA PONCOVÁ

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

HLUBOKÉ ZÁSOBNÍKOVÉ AUTOMATY KONEČNÉHO INDEXU

DEEP PUSHDOWN AUTOMATA OF FINITE INDEX

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VENDULA PONCOVÁ

VEDOUCÍ PRÁCE
SUPERVISOR

prof. RNDr. ALEXANDER MEDUNA, CSc.

BRNO 2013

Abstrakt

Tato práce představuje několik modifikací hlubokých zásobníkových automatů s ohledem na redukci počtu stavů nebo nevstupních symbolů. Je ukázáno, že síla hlubokých zásobníkových automatů konečného indexu není ovlivněna omezením nevstupních symbolů na jeden, tudíž tyto automaty charakterizují nekonečnou hierarchii jazykových rodin vycházejících z programových gramatik konečného indexu. Na základě principu tohoto automatu je stanovena normální forma hlubokých zásobníkových automatů. Nakonec zavádím zobecněný hluboký zásobníkový automat, který expanduje nejvrchnější možný nevstupní symbol na zásobníku. Tento automat spolu s jeho zredukovanými formami je ekvivalentní se stavovými gramatikami.

Abstract

This thesis introduces several modifications of deep pushdown automata considering the reduced number of states or non-input symbols. It is shown that the power of deep pushdown automata of finite index is not affected by a limitation of non-input symbols to one, thus these automata characterize an infinite hierarchy of language families resulting from programmed grammars of finite index. Based on a principle of these automata, it is established the normal form of deep pushdown automata. Finally, I introduce generalized deep pushdown automata. They expand the topmost possible non-input symbol in the pushdown. These automata and their reduced forms are equivalent to state grammars.

Klíčová slova

stavová gramatika, programová gramatika, hluboký zásobníkový automat, redukce stavů, redukce nevstupních symbolů, normální forma

Keywords

state grammar, programmed grammar, deep pushdown automata, reduction of states, reduction of non-input symbols, normal form

Citace

Vendula Poncová: Hluboké zásobníkové automaty konečného indexu, bakalářská práce, Brno, FIT VUT v Brně, 2013

Hluboké zásobníkové automaty konečného indexu

Prohlášení

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

.....
Vendula Poncová
14. května 2013

Poděkování

Tímto způsobem bych ráda poděkovala vedoucímu své práce, panu prof. RNDr. Alexandru Medunovi, CSc., za věnovaný čas a pomoc při řešení.

Obsah

Kapitola 1

Úvod

Od vzniku teorie formálních jazyků byl kladen důraz na bezkontextové jazyky. Ve většině případů jsou jejich možnosti postačující, a proto je dnes tato problematika velmi dobře prozkoumaná a zdokumentovaná. Nicméně „svět je kontextový“ [1]. V oblastech jako jsou přirozené jazyky, vývojová biologie nebo například sémantika programovacích jazyků je kontext důležitý. Na druhou stanu kontextové a vyšší jazyky jsou příliš mocné a špatně se s nimi pracuje. Řešením bylo spojit „jednoduchost a krásu bezkontextových gramatik se sílou kontextových“ [1] a zavést tzv. řízené gramatiky. Mnohé z těchto gramatik definují jazykovou rodinu ležící mezi rodinou bezkontextových jazyků a jazyků kontextových.

V důsledku tohoto vývoje se začaly zavádět automaty ekvivalentní těmto gramatikám. Příkladem budiž hluboký zásobníkový automat [6] korespondující s n-limited stavovou gramatikou a hluboký zásobníkový automat konečného indexu [7] ekvivalentní s maticovou gramatikou konečného indexu. V této práci zkoumám tyto dva modely a zavádím jejich nové modifikace. Hlavním kritériem je redukce. Zaměřuji se na snížení počtu stavů, nevstupních symbolů, případně zjednodušení pravidel a zkoumám dopad na sílu automatu.

V kapitole 3 zavádím hluboký zásobníkový automat konečného indexu s jedním nevstupním symbolem a dokazuji jeho ekvivalenci s programovými gramatikami konečného indexu. Způsob, jakým tento model pracuje, aplikuji na hluboký zásobníkový automat v kapitole 4. Výsledkem je zavedení normální formy hlubokého zásobníkového automatu a algoritmus pro převod automatů do této formy. V následující kapitole 5 představuji zobecněný hluboký zásobníkový automat. Zobecnění spočívá v expanzi nejvrchnějšího symbolu na zásobníku, který lze v daném stavu expandovat. Původní hluboký zásobníkový automat měl hloubku expanze definovanou v pravidle. Dále zkoumám sílu zobecněného automatu vzhledem ke stavovým gramatikám a pokouším se minimalizovat počet nevstupních symbolů a stavů. Způsob této redukce demonstruji v aplikaci, jejíž návrh a implementaci popisuji v kapitole 6. Jedná se o konzolovou aplikaci, která na standardní výstup vypisuje zredukovaný zobecněný zásobníkový automat dle nastavených parametrů, případně derivační kroky pro přijetí zadaného řetězce.

Kapitola 2

Pojmy a definice

V této kapitole zavádím a definuji pojmy, ze kterých vycházím v dalších částech své práce. Ve stručnosti se zabývám problematikou řízených gramatik a definuji programové a stavové gramatiky. Dále uvádím definici hlubokého zásobníkového automatu a jeho konečné varianty. Nakonec zařazuji jazyky přijímané těmito automaty spolu s jazyky generovanými řízenými gramatikami do kontextu označovaného jako hierarchie mezi kontextovými a bezkontextovými jazyky.

2.1 Základní pojmy

V této práci předpokládám základní znalosti z teorie formálních jazyků. Dále zavádím následující konvence.

Abeceda je konečná neprázdná množina symbolů. *Prázdný řetězec* se značí symbolem ε. Nechť Σ je abeceda, pak ε je *řetězec* nad abecedou Σ, a pokud x je řetězec nad Σ a $a \in \Sigma$, pak xa je řetězec nad abecedou Σ. Σ^* je množina všech řetězců nad Σ a $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. Pro řetězec $v \in \Sigma$ platí, že $|v|$ označuje délku řetězce v a $\mathrm{alph}(v)$ značí množinu symbolů vyskytujících se v řetězci v. Pokud V je množina symbolů, pak $\mathrm{occur}(v, V)$ značí počet výskytů symbolů z V v řetězci v. Nechť I je podmnožina celých čísel, pak $\max(I)$ označuje největší číslo v I.

2.2 Hluboké zásobníkové automaty

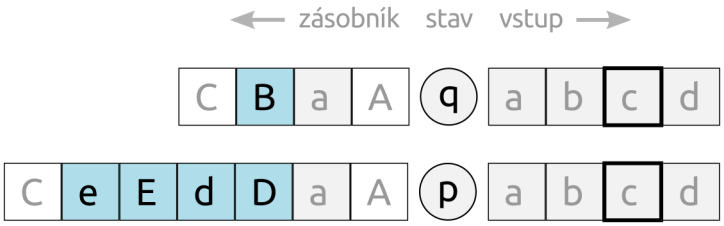
Zásobníkový automat je modelem pro bezkontextové jazyky. Umožňuje přepisovat symbol na vrcholu zásobníku řetězcem. A. Meduna tento automat zobecnil a zvýšil jeho sílu tak, že lze přepisovat libovolný symbol na zásobníku. Tyto automaty se označují jako *hluboké zásobníkové automaty*.

Definice 2.2.1. *Hluboký zásobníkový automat* je dle [6] definován jako uspořádaná sedmice $M = (Q, \Sigma, \Gamma, R, s, S, F)$, kde

Q je konečná množina stavů,
Σ vstupní abeceda,
Γ zásobníková abeceda, přičemž $\Sigma \subseteq \Gamma$ a $Q \cap \Gamma = \emptyset$,
R je konečná množina pravidel typu $(m, q, A, p, v) \in (\{1, 2, 3, \dots\} \times Q \times (\Gamma - \Sigma) \times Q \times \Gamma^+)$, píšeme $mqA \to pv$,
$s \in Q$ je počáteční stav,

$S \in \Gamma$ počáteční zásobníkový symbol,
$F \subseteq Q$ je množina koncových stavů.

Konfigurace automatu M je prvek z množiny $(Q \times \Sigma^* \times \Gamma^*)$. Nechť X, Y jsou dvě konfigurace. M přečte symbol na vstupní pásce a přejde z X do Y, tzv. *pop operace*, píšeme $X \ {}_p\Rightarrow Y \ [mqA \to pv]$, zjednodušeně $X \ {}_p\Rightarrow Y$, pokud $X = (q, au, az)$, $Y = (q, u, z)$, kde $q \in Q$, $a \in \Sigma$, $u \in \Sigma^*$, $z \in \Gamma^*$. M přepíše symbol na zásobníku a přejde z X do Y, tzv. *expanze*, píšeme $X \ {}_e\Rightarrow Y \ [mqA \to pv]$, zjednodušeně $X \ {}_e\Rightarrow Y$, pokud $X = (q, w, uAz)$, $Y = (p, w, uvz)$, $mqA \to pv \in R$ a platí $\text{occur}(u, \Gamma - \Sigma) = m - 1$, kde p, $q \in Q$, $w \in \Sigma^*$, $A \in \Gamma$, $u, v, z \in \Gamma^*$. M provede *přechod* z X do Y, píšeme $X \Rightarrow Y \ [mqA \to pv]$, zjednodušeně $X \Rightarrow Y$, pokud provede expanzi nebo operaci pop.

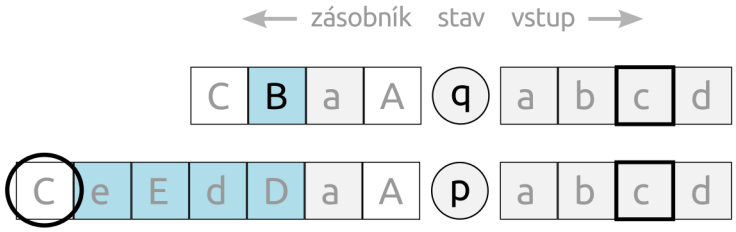


Obrázek 2.1: Aplikace pravidla $2qB \to pDdEe$ v hlubokém zásobníkovém automatu .

Nechť X je konfigurace. M provede *nula přechodů* z X do X, píšeme $X \Rightarrow^0 X[\varepsilon]$, zjednodušeně $X \Rightarrow^0 X$. Nechť $X_0, X_1, X_2, \ldots, X_n$ je sekvence přechodů konfigurací pro $n \geq 1$ a $X_{i-1} \Rightarrow X_i[r_i]$, kde $r_i \in R$, pro každé $i \in \{1, 2, 3, \ldots, n\}$, tedy platí $X_0 \Rightarrow X_1[r_1] \Rightarrow X_2[r_2] \cdots \Rightarrow X_n[r_n]$. Pak M provede n *přechodů* z X_0 do X_n. Píšeme $X_0 \Rightarrow^n X_n[r_1 \ldots r_n]$, zjednodušeně $X_0 \Rightarrow^n X_n$. Pokud $X_0 \Rightarrow^n X_n[r_1 \ldots r_n]$ pro nějaké $n \geq 1$, pak $X_0 \Rightarrow^+ X_n[r_1 \ldots r_n]$. Pokud $X_0 \Rightarrow^n X_n[r_1 \ldots r_n]$ pro nějaké $n \geq 0$, pak $X_0 \Rightarrow^* X_n[r_1 \ldots r_n]$.

Říkáme, že pravidlo $mqA \to pv \in R$ je hloubky m. Pokud existuje nejmenší přirozené číslo n takové, že každé pravidlo v M je hloubky menší nebo rovno n, říkáme, že M je hloubky n, píšeme ${}_nM$. Jazyk přijímaný automatem ${}_nM$ je

$$L({}_nM) = \{w \in \Sigma^* \mid (s, w, S) \Rightarrow^* (f, \varepsilon, \varepsilon), \text{ kde } f \in F\}.$$



Obrázek 2.2: V hlubokém zásobníkovém automatu indexu 3 nelze pravidlo $2qB \to pDdEe$ v aktuální konfiguraci aplikovat.

Definice 2.2.2. Podle [7] je *hluboký zásobníkový automat konečného indexu* uspořádaná osmice $M_n = (Q, \Sigma, \Gamma, R, s, S, F, n)$, jehož definice vychází z hlubokého zásobníkového automatu. Symbol $n \in \{1, 2, 3, \ldots\}$ označuje maximální počet nevstupních symbolů, které

mohou být uložené na zásobníku. Expanze $X \ {}_e\Rightarrow Y$ může proběhnout jen v případě, že v konfiguraci Y bude na zásobníku n a méně nevstupních symbolů. O hloubce zásobníku tohoto typu automatu se předpokládá, že je rozšiřitelná tak, aby zásobník mohl přijmout až n nevstupních symbolů.

2.3 Řízené gramatiky

Řízené gramatiky vycházejí z definice bezkontextové gramatiky. Zachovává se základní tvar pravidla, ale mění se způsob derivace. Tímto způsobem lze zvýšit mocnost modelu bez zbytečného komplikování pravidel. Mezi řízené gramatiky patří například programové, maticové a stavové.

Definice 2.3.1. *Bezkontextová gramatika* [4] je čtveřice $G = (V, T, P, S)$, kde

$V = T \cup N$ je úplná abeceda,
T je abeceda terminálů,
N je abeceda neterminálů,
$S \in N$ je počáteční symbol,
P je konečná množina pravidel tvaru $A \to v$, kde $A \in N$, $v \in V^*$.

Nechť $A \to v \ \in P$, $A \in N$, v, x, $y \in V^*$, pak bezkontextová gramatika G provede derivační krok z xAy do xvy, píšeme $xAy \Rightarrow xvy[A \ \to v]$, zjednodušeně $xAy \Rightarrow xvy$. Nechť $r_1, r_2, \ldots r_m \in P$ pro $m \geq 0$, pak G může provést sekvenci kroků dle těchto pravidel, zápisem $x \Rightarrow^m y[r_1 r_2 \ldots r_m]$. Píšeme $\Rightarrow^+$ pro libovolné $m > 0$ a $\Rightarrow^*$ pro $m \geq 0$.

Bezkontextová gramatika G generuje jazyk $L(G)$, pro který platí

$$L(G) = \{w \in T^* | S \Rightarrow^* w\}.$$

Definice 2.3.2. *Programová gramatika* [4] je čtveřice $G = (V, T, P, S)$, kde V, T a S jsou definované stejně jako v bezkontextových gramatikách. P je konečná množina pravidel tvaru $r\colon A \ \to v, g(r)$, kde r je označení pravidla, $A \to v$ je pravidlo bezkontextové gramatiky a $g(r)$ je množina značení těch pravidel, která mohou být provedena v dalším derivačním kroku po aplikaci pravidla r.

Definice 2.3.3. *Programová gramatika konečného indexu n* [4] je programová gramatika $G = (V, T, P, S)$, pro jejíž každou větnou formu $w \in L(G)$ existuje taková posloupnost derivačních kroků, kde se v žádném kroku nevyskytuje více než n neterminálů.

Definice 2.3.4. *Stavová gramatika* je dle [3] šestice $G = (V, W, T, S, s, P)$, kde

$V = T \cup N$ je úplná abeceda,
W je konečná množina stavů,
T je abeceda terminálů,
N je abeceda neterminálů,
$S \in N$ je počáteční symbol,
$s \in W$ je počáteční stav,
P je konečná relace $(W \times N) \times (W \times V^+)$, pravidla zapisujeme $(q, A) \to (p, v)$.

Nechť $(q, A) \to (p, v) \in P$, $p, q \in W$, $A \in N$, $v, x, y \in V^*$, pak stavová gramatika G provede derivační krok z xAy do xvy, píšeme $(q, xAy) \Rightarrow (p, xvy)[(q, A) \to (p, v)]$, zjednodušeně $xAy \Rightarrow xvy$, právě tehdy, když

$$\{Z \mid (q, Z) \to (p', v') \in P,\ Z \ \in \mathrm{alph}(x) \cap N,\ p' \in W,\ v' \in V^*\} = \emptyset.$$

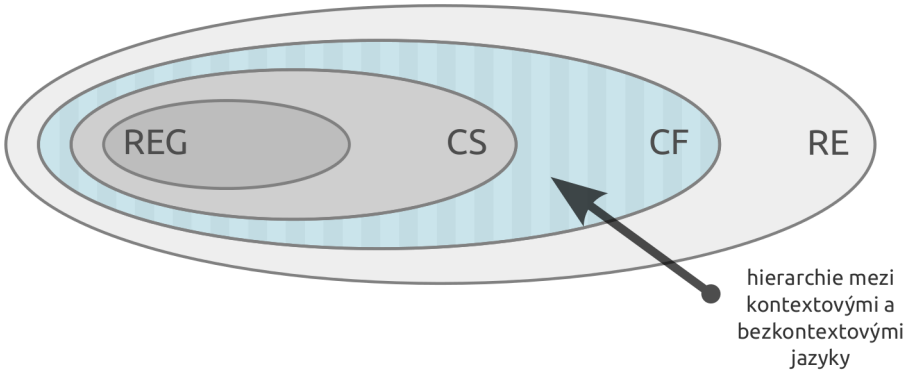
Gramatika G generuje jazyk

$$L(G) = \{w \in T^* | (s, S) \Rightarrow^* (p, w), p, q \in W\}.$$

Definice 2.3.5. *Stavová gramatika konečného indexu n* [3] je stavová gramatika $G = (V, W, T, S, s, P)$, pro kterou platí, že derivační krok $(q, xAy) \Rightarrow (p, xvy)$ za pomoci pravidla $(q, A) \rightarrow (p, v) \in P$, kde p, $q \in W$, $A \in N$, v, x, $y \in V^*$, lze provést právě tehdy, když je počet neterminálů v řetězci xA menší nebo roven n. Píšeme ${}_n\Rightarrow$.

Definice 2.3.6. *Stavová gramatika s ε-pravidly* je podle [2] stavová gramatika $G = (V, W, T, S, s, P)$ rozšířená o vymazávací pravidla typu $(q, A) \rightarrow (p, \varepsilon)$, kde p, $q \in W$, $A \in N$.

2.4 Hierarchie mezi kontextovými a bezkontextovými jazyky

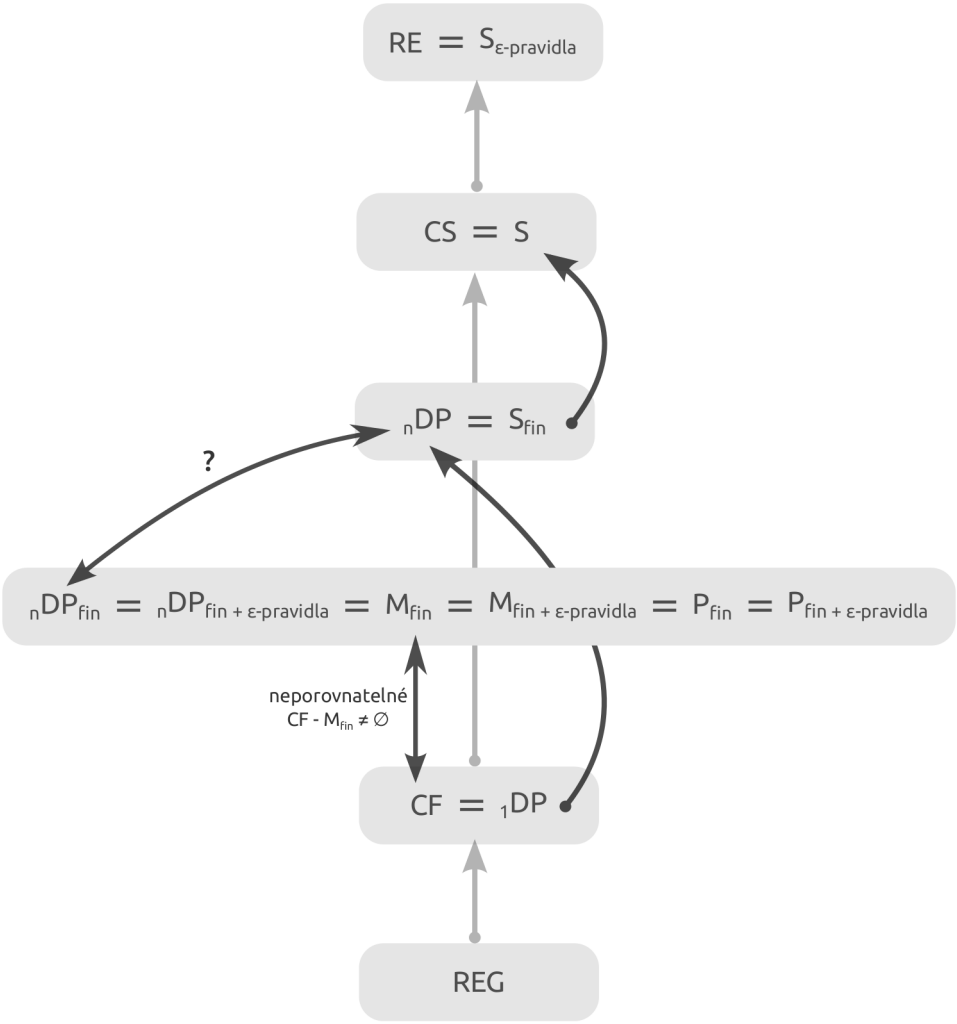


Obrázek 2.3: Znázornění zkoumané oblasti v kontextu Chomského hierarchie.

Mocnosti definovaných modelů se v rámci Chomského hierarchie nacházejí přibližně mezi kontextovými a bezkontextovými jazyky (obrázek 2.3). Pro bližší zkoumání vztahů mezi jazykovými rodinami zavedu následující označení:

RE je rodina rekurzivně spočetných jazyků,
CF je rodina kontextových jazyků,
CS je rodina bezkontextových jazyků,
REG je rodina regulárních jazyků,
${}_n$**DP** je rodina jazyků přijímaných hlubokým zásobníkovým automatem hloubky $n \geq 1$,
M je rodina jazyků generovaných maticovými gramatikami,
P je rodina jazyků generovaných programovými gramatikami,
S je rodina jazyků generovaných stavovými gramatikami.

X_{fin} indikuje konečný index modelu X. Pro modely jazykových rodin **M**, **P** a **S** platí, že vycházejí z bezkontextové gramatiky a neobsahují ε-pravidla. Vztahy mezi jazykovými rodinami jsou znázorněné na diagramu 2.4. Vycházela jsem z výsledků v pracích A. Meduny [6, 7, 2], J. Dassow [1] a T. Kasai [3].



Obrázek 2.4: Grafické srovnání mocností jazykových rodin některých modelů.

Kapitola 3

Hluboký zásobníkový automat s jedním nevstupním symbolem

V této kapitole se zabývám otázkou, jaký je vliv počtu nevstupních symbolů na mocnost hlubokého zásobníkového automatu konečného indexu. V souvislosti s touto problematikou představuji hluboký zásobníkový automat s jedním nevstupním symbolem a porovnávám jej s programovými gramatikami konečného indexu.

3.1 Omezení počtu nevstupních symbolů

Omezím-li počet nevstupních symbolů hlubokého zásobníkového automatu konečného indexu M na jeden, mohu definovat nový typ automatu $M_{\#}$. Jestliže chci zachovat sílu automatu M, musím být schopna jeho činnost simulovat pomocí $M_{\#}$. Pro tyto účely zavádím stavy, kde uchovávám informace o stavu automatu M a o nevstupních symbolech na zásobníku. To je možné díky konečnému počtu těchto symbolů na zásobníku, neboť jinak by takové řešení vedlo k nekonečnému počtu stavů. Každé pravidlo pak lze pro každý možný obsah zásobníku převést na jeho ekvivalent nahrazením nevstupních symbolů symbolem # a jejich uložením do stavu. Převod automatu M na $M_{\#}$ popisuji algoritmem 1.

Algoritmus 1. Převod hlubokého zásobníkového automatu konečného indexu na ekvivalentní s jedním nevstupním symbolem.

Vstup: $M = (Q, \Sigma, \Gamma, R, s, S, F, n)$
Výstup: $M_{\#} = (Q_{\#}, \Sigma_{\#}, \Gamma_{\#}, R_{\#}, s_{\#}, S_{\#}, F_{\#}, n)$

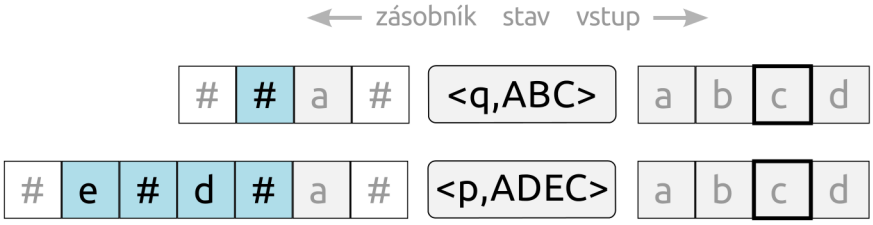
$\Sigma_{\#} := \Sigma$
$\Gamma_{\#} := \{\#\} \cup \Sigma$
$s_{\#} :=< s, \# >$
$S_{\#} := \#$

Pro každé $mqA \to pv \in R$, kde $v = b_0 B_1 b_1 B_2 b_2 \dots b_{j-1} B_j b_j$, $j \in \{0, 1, 2, \dots, n-1, n\}$, $b_0, b_i \in \Sigma^*$ a $B_i \in (\Gamma - \Sigma)$ pro všechna $i \in \{1, 2, \dots, j\}$:

A pro každé $(u, z) \in (\Gamma - \Sigma)^* \times (\Gamma - \Sigma)^*$, kde $|u| = m - 1$, $|z| \leq n - m$:

přidej do $Q_{\#}$ stavy $< q, u\ A\ z >$, $< p, u\ B_1 B_2 \dots B_{j-1} B_j z >$,
přidej do $R_{\#}$ $m < q, u\, A\, z > \# \to < p, u\, B_1 B_2 \dots B_{j-1} B_j z > b_0 \# b_1 \# b_2 \dots b_{j-1} \# b_j$,
pokud $q \in F$, přidej do $F_{\#}$ stav $< q, u\ A\ z >$,
pokud $p \in F$, přidej do $F_{\#}$ stav $< p, u\ B_1 B_2 \dots B_{j-1} B_j z >$.

Je zřejmé, že každý hluboký zásobníkový automat konečného indexu s jedním nevstupním symbolem splňuje definici pro obecný hluboký zásobníkový automat konečného indexu. Tudíž spolu s algoritmem 1 jsem neformálně dokázala, že tyto automaty jsou ekvivalentní.



Obrázek 3.1: Aplikace pravidla $2qB \rightarrow pDdEe$ upraveného podle algoritmu 1.

3.2 Ekvivalence s programovými gramatikami konečného indexu

A. Meduna [7] dokázal, že hluboké zásobníkové automaty konečného indexu jsou ekvivalentní s maticovými gramatikami konečného indexu, tudíž třídy jazyků, které zásobníkový automat přijímá, tvoří nekonečnou hierarchii. Vzhledem k výsledkům v kapitole 3.1 lze očekávat, že tato vlastnost automatu zůstane i při omezení počtu nevstupních symbolů. Konstrukci důkazu tohoto tvrzení ukážu na ekvivalenci s programovými gramatikami, přičemž platí, že maticové a programové gramatiky konečného indexu generují stejný jazyk [1].

Pro hluboký zásobníkový automat podle článku [7] platí, že jestliže přijme nějaké slovo w, pak existuje taková sekvence přechodů přijímající slovo w, která provádí pouze pop operace po první pop operaci. Toto tvrzení využiji v následujícím důkazu, neboť zásobníkový automat, na kterém probíhají jen expanze, funguje jako gramatika. Stačí proto ukázat, že automat je schopen na svém zásobníku simulovat všechny derivace gramatiky a gramatika je schopná simulovat automat.

V algoritmu 2 popisuji konstrukci hlubokého zásobníkového automatu konečného indexu s jedním nevstupním symbolem M simulujícího programovou gramatiku konečného indexu G. Stav automatu se v tomto případě skládá ze dvou položek: označení pravidla, které se bude simulovat v dalším kroku, a řetězce neterminálů, které jsou na zásobníku nahrazené symbolem #. Automat přejde do koncového stavu, pokud jeho zásobník neobsahuje žádné nevstupní symboly.

Algoritmus 2. Převod programové gramatiky konečného indexu na ekvivalentní hluboký zásobníkový automat konečného indexu s jedním nevstupním symbolem.

Vstup: $G = (T \cup N, T, P, S)$ konečného indexu n
Výstup: $M = (Q, \Sigma, \Gamma, R, s, S\ , F, n)$

$\Sigma := T$
$\Gamma := T \cup \{\#\}$
$s_{\#} :=< \sigma >$
$S_{\#} := \#$

Pro každé $p : S \ \rightarrow v, g(p) \in P$:
přidej do R pravidlo $< \sigma >_1 \# \rightarrow < p, S > \#$ a do Q stav $< p, S >$.

Pro každé $q \in (Q \cup \{\varepsilon\})$:
přidej do F stav $< q, \varepsilon >$.

Pro každé $p : A \ \rightarrow v, g(p) \in P$, kde $v = b_0 B_1 b_1 B_2 b_2 \ldots b_{j-1} B_j b_j$, $j \in \{0, 1, 2, 3, \ldots, n\}$, $b_0, b_i \in T^*$ a $B_i \in N$ pro všechna $i \in \{1, 2, \ldots, j\}$:

Pro každé $(k, u, z) \in \{1, 2, 3, \ldots, n - j + 1\} \times N^* \times N^*$, kde $|u| = k - 1$, $|z| \leq n - k$:

Pokud $g(p) = \emptyset$:
přidej do Q stavy $< p, uAz >$, $< \varepsilon, u \ B_1 B_2 \ldots B_{j-1} B_j z >$,
přidej do R pravidlo :
$< p, uAz >_k \# \rightarrow < \varepsilon, u \ B_1 B_2 \ldots B_{j-1} B_j z > b_0 \# b_1 \# b_2 \ldots b_{j-1} \# b_j$.

Jinak pro každé $q \in g(p)$:
přidej do Q stavy $< p, uAz >$, $< q, u \ B_1 B_2 \ldots B_{j-1} B_j z >$,
přidej do R pravidlo :
$< p, uAz >_k \# \rightarrow < q, u \ B_1 B_2 \ldots B_{j-1} B_j z > b_0 \# b_1 \# b_2 \ldots b_{j-1} \# b_j$.

Pro konstrukci důkazu o převodu hlubokého zásobníkového automatu konečného indexu s jedním nevstupním symbolem na programovou gramatiku konečného indexu lze beze změny použít postup z článku [4], který srovnává programové gramatiky s #-Rewriting Systems. Programová gramatika simuluje každý krok zásobníkového automatu sekvencí několika derivací. Neterminály obsahují informace o stavu zásobníku p, o aktuální pozici výskytu symbolu $\# \ i$ a celkovém počtu symbolů $\# \ h$ ve formě zápisu $< p, i, h >$. Simulace aplikace pravidla s hloubkou expanze n probíhá následovně:

1. Ve všech neterminálech se aktualizuje jejich pozice a celkový počet symbolů # vzhledem k výsledné konfiguraci simulovaného automatu. Symboly se expandují postupně zleva doprava tak, že po každé expanzi se vybere pravidlo pro následující neterminál.

2. Podle simulovaného pravidla se expanduje neterminál na pozici n. Pozice neterminálů za tímto symbolem byly v kroku 1 upraveny tak, aby nové neterminály vyplnily chybějící pozice.

3. Neterminály se postupně prochází zleva doprava a pomocný stav je přepisován stavem aktuálním. Tím je simulace dokončena.

Ukázala jsem, že výše zavedený typ zásobníkového automatu je ekvivalentní s programovými gramatikami konečného indexu. Z toho vyplývá, že rodina jazyků přijímaná tímto automatem tvoří nekonečnou hierarchii vycházející z konečných programových gramatik.

Kapitola 4

Hluboký zásobníkový automat v normální formě

V této kapitole zavádím normální formu hlubokého zásobníkového automatu a ukazuji způsob, jakým lze automat na tuto formu převést. Vycházím z myšlenky automatu s jedním nevstupním symbolem, kterému jsem se věnovala doposud. Hluboký zásobníkový automat je ale zobecněním konečného a nemá omezení pro počet nevstupních symbolů na zásobníku. Z toho důvodu není možné na něj aplikovat algoritmus 1, neboť by to vedlo k nekonečnému počtu stavů a pravidel. Je však možné pracovat podobným způsobem s tzv. prefixem zásobníku, kdy nejvrchnějších n nevstupních symbolů nahrazuji symbolem #. Celkový počet nevstupních symbolů jsem tak nijak nezredukovala, ale docílila jsem zjednodušeného zápisu pravidel.

4.1 Definice normální formy

Hluboký zásobníkový automat v normální formě dle definice 4.1.1 pracuje s prefixem n nevstupních symbolů. Přepis symbolů v prefixu na symbol # umožňuje pravidlo typu (i), změnu stavu pravidlo typu (ii), přepis na symbol vstupní abecedy pravidlo typu (iii) a expandovat lze pravidlem typu (iv).

Definice 4.1.1. Nechť $M = (Q, \Sigma, \Gamma, R, s, S\ , F)$ je hluboký zásobníkový automat a n je maximální hloubka expanze v M. Pak M je v normální formě, pokud každé pravidlo z R je v jednom ze tvarů:

(i) $mqA \to q\#$ pro každé m, q, $A : 1 \leq m \leq n$, $q \in Q$, $A \in (\Gamma - \Sigma) - \{\#\}$,

(ii) $mq\# \to p\#$, kde $1 \leq m \leq n$, p, $q \in Q$,

(iii) $mq\# \to pa$, kde $a \in \Sigma^+$, $1 \leq m \leq n$, p, $q \in Q$,

(iv) $mq\# \to p\#A$, kde $A \in (\Gamma - \Sigma) - \{\#\}$, $1 \leq m \leq n$, p, $q \in Q$.

Nechť pravidlo typu (i) lze aplikovat jen na nejvrchnější možný nevstupní symbol na zásobníku a typ (iv) je možné aplikovat jen na nejspodnější symbol #, pak na zásobníku je v každém kroku řetězec z množiny $(\Sigma^*\{\#\})^k\Gamma^*$, kde $0 \leq k\ \leq n$.

Obrázek 4.1: Příklad konfigurace hlubokého zásobníkového automatu v normální formě s prefixem 3.

4.2 Převod na normální formu

Při sestavování algoritmu pro převod hlubokého zásobníkového automatu do normální formy jsem musela řešit zejména omezené možnosti expanze. Původní stav automatu a nevstupní symboly z prefixu opět uchovávám ve stavech. Pokud bych ale připustila expanzi na libovolný počet nevstupních symbolů, ztratila bych ze stavu informaci o nahrazených nevstupních symbolech, které by se ocitly za prefixem, nebo bych zvětšovala velikost prefixu a počet pomocných stavů by tak narostl do nekonečna. Tento problém jsem vyřešila tzv. postupnou expanzí [5]. Řetězec na pravé straně pravidla je reprezentován jedním nevstupním symbolem. Ten je možné expandovat na právě dva nevstupní symboly: první symbol v řetězci a zbývající část řetězce. Jednoduchou rekurzí tohoto postupu lze dosáhnout úplné expanze.

Expanze na dva nevstupní symboly je již udržitelná, ale pokud bych neomezila použití pravidla typu (iv), stále by se mohlo stát, že poslední symbol # bude odsunut o jednu pozici za prefix. Nabízelo se povolit pravidlo pro přepis tohoto symbolu na původní nevstupní symbol, ale tím bych zvýšila hloubku automatu. Místo toho spojuji krok expanze a krok přepisu posledního # symbolu na původní symbol do jednoho pravidla. Expanze tak nemá vliv na # symboly v prefixu.

V algoritmu 3 popisuji konstrukci hlubokého zásobníkového automatu v normální formě. S ohledem na postupnou expanzi zavádím nové nevstupní symboly, jejichž součástí je rovněž původní pravidlo pro expanzi. Nelze určit dopředu, kdy se jednotlivé kroky expanze provedou, proto je nutné umožnit je provést ve všech stavech automatu a na všech pozicích v prefixu. Ukázka aplikace tohoto algoritmu je k dispozici v příkladu 4.2.1.

Algoritmus 3. Převod hlubokého zásobníkového automatu do normální formy.

Vstup: $M = (Q, \Sigma, \Gamma, R, s, S, F)$
Výstup: $M_{NF} = (Q_{NF}, \Sigma_{NF}, \Gamma_{NF}, R_{NF}, s_{NF}, S_{NF}, F_{NF})$

$\Sigma_{NF} := \Sigma$
$s_{NF} :=< s, S >$
$S_{NF} := \#$

$k := \max(\{m \mid mqA \to pv \in R\})$

Pro každé $(q, u, A) \in Q \times (\Gamma - \Sigma)^* \times (\Gamma - \Sigma)$, kde $|u| < k$:

přidej do R_{NF} pravidlo typu (i) $|uA| < q, u > A \ \to < q, uA > \#$.

Pro každé $r : mqA \to pX_1X_2X_3 \ldots X_j \in R$, kde $1 \leq i \leq j$, $X_i \in \Gamma$, a pro každou dvojici $(u, v) \in (\Gamma - \Sigma)^* \times (\Gamma - \Sigma)^*$, kde $|uv| < k$:

Pokud $|u| = m - 1$, $|v| \leq k - m$, přidej do R_{NF} pravidlo typu (ii) :

$m < q, uAv > \# \rightarrow < p, u(r : X_1X_2X_3 \dots X_j)v > \#$.

Pro každé $(X_i, q') \in \{X_1, X_2, X_3, \dots, X_j\} \times Q$:

Pokud $X_i \in (\Gamma - \Sigma)$, přidej do R_{NF} pravidlo typu (ii) :
$|u| + 1 < q', u(r : X_i)v > \# \rightarrow < q', uX_iv > \#$.

Pokud $X_i \in \Sigma$, přidej do R_{NF} pravidlo typu (iii) :
$|u| + 1 < q', u(r : X_i)v > \# \rightarrow < q', uv > X_i$.

Nechť $v = V_1V_2V_3 \dots V_{l-1}V_l$, kde V_1, V_2, V_3, …, V_{l-1}, $V_l \in (\Gamma - \Sigma)$.
Pak pokud $i < j$, přidej do R_{NF} pravidlo typu (iv) :

$|uv| + 1 < q', u(r : X_iX_{i+1} \dots X_j)v > \# \rightarrow < q', u(r : X_i)(r : X_{i+1} \dots X_j)v' > \#V'$,
kde $v \neq \varepsilon$, $v' = V_1V_2V_3 \dots V_{l-1}$ a $V' = V_l$, jinak

$|u| + 1 < q', u(r : X_iX_{i+1} \dots X_j) > \# \rightarrow < q', u(r : X_i) > \#(r : X_{i+1} \dots X_j)$.

$\Gamma_{NF} := \{\#\} \cup \Sigma \cup \{A, X \mid mqA \rightarrow pv \in R_{NF}, X \in \text{alph}(v) \cap (\Gamma - \Sigma)\}$
$Q_{NF} := \{p, q \mid mqA \rightarrow pv \in R_{NF}\}$
$F_{NF} := \{< q, \varepsilon > \mid q \in F\}$

Každý hluboký zásobníkový automat lze převést na ekvivalentní automat v normální formě. Zároveň je možné podobným způsobem modifikovat každý automat v normální formě tak, že ve stavech zaznamenávám symboly # vyskytující se v prefixu zásobníku a s jejich pomocí simuluji použití pravidel (i) a (iv). Pak omezení kladená na tato pravidla nejsou nutná a takový automat splňuje definici hlubokého zásobníkového automatu. Z toho vyplývá, že se jedná o ekvivalentní modely.

Příklad 4.2.1. M je hluboký zásobníkový automat s maximální hloubkou expanze 3 a M_{NF} je jeho normální forma zkonstruovaná podle algoritmu 3. Pro lepší přehlednost definice automatů nejsou kompletní a pravidla jsou označena identifikátorem.

$$
\begin{aligned}
M = (\ & \{s, p, q, \ldots, f\},\\
& \{a, b, c, d, e\},\\
& \{a, b, c, d, e, A, B, C, D, E, S\},\\
& \{\ \mathrm{r}:\ \ 2qB \to pDdEe, \ldots\},\\
& s,\\
& S,\\
& \{\mathrm{f}\}\ )
\end{aligned}
$$

$$
\begin{aligned}
M_{NF} = (\ & \{\ < s, S >, < q, ABC >, < p, A(r : DdEe)C >, < p, A(r : D)(r : dEe) >,\\
& \quad < p, AD(r : dEe) >, < p, AD(r : d) >, < p, AD >, < p, AD(r : Ee) >,\\
& \quad < p, AD(r : E) >, < p, ADE >, \ldots, < f, \varepsilon >\\
& \},\\
& \{\ a, b, c, d, e\ \},\\
& \{\ a, b, c, d, e, A, B, C, D, E, S, (r : DdEe), (r : D), (r : dEe), (r : d),\\
& \quad (r : Ee), (r : E), (r : e), \ldots, \#\\
& \},\\
& \{\\
& \quad 1:\ \ 2\ < q, ABC >\ \#\ \to < p, A(r : DdEe)C >\ \#,\\
& \quad 2:\ \ 2\ < p, A(r : DdEe)C >\ \#\ \to < p, A(r : D)(r : dEe) >\ \#,\\
& \quad 3:\ \ 2\ < p, A(r : D)(r : dEe) >\ \#\ \to < p, AD(r : dEe) >\ \#,\\
& \quad 4:\ \ 3\ < p, AD(r : dEe) > \#\ \to < p, AD(r : d) > (r : Ee),\\
& \quad 5:\ \ 3\ < p, AD(r : d) >\ \#\ \to < p, AD >\ d,\\
& \quad 6:\ \ 3\ < p, AD >\ (r : Ee)\ \to < p, AD(r : Ee) >\ \#,\\
& \quad 7:\ \ 3\ < p, AD(r : Ee) >\ \#\ \to < p, AD(r : E) >\ (r : e),\\
& \quad 8:\ \ 3\ < p, AD(r : E) >\ \#\ \to < p, ADE >\ \#,\\
& \quad \ldots\\
& \},\\
& < s, S >,\\
& \#,\\
& \{\ < f, \varepsilon >\ \}\ )
\end{aligned}
$$

Ukázka simulace pravidla $r : 2qB \to pDdEe$ automatu M pro konfiguraci $(q, abc, AaBC)$ v hlubokém zásobníkovém automatu v normální formě M_{NF} s prefixem 3.

	$(< q, ABC >,$	$abc,$	$\#a\#\#\ )$		
$\Rightarrow$	$(< p, A(\mathbf{r} : \mathbf{DdEe})C >,$	$abc,$	$\#a\#\#\ )$	[1]	postupná expanze podle r
$\Rightarrow$	$(< p, A(\mathbf{r} : \mathbf{D})(\mathbf{r} : \mathbf{dEe}) >,$	$abc,$	$\#a\#\#\mathbf{C}\ )$	[2]	
$\Rightarrow$	$(< p, A\mathbf{D}(r : dEe) >,$	$abc,$	$\#a\#\#C\ )$	[3]	expanze na nevstupní symbol
$\Rightarrow$	$(< p, AD(\mathbf{r} : \mathbf{d}) >,$	$abc,$	$\#a\#\#(\mathbf{r} : \mathbf{Ee})C\ )$	[4]	
$\Rightarrow$	$(< p, AD >,$	$abc,$	$\#a\#\mathbf{d}(r : Ee)C\ )$	[5]	expanze na vstupní symbol
$\Rightarrow$	$(< p, AD(\mathbf{r} : \mathbf{Ee}) >,$	$abc,$	$\#a\#d\#C\ )$	[6]	načtení symbolu do prefixu
$\Rightarrow$	$(< p, AD(\mathbf{r} : \mathbf{E}) >,$	$abc,$	$\#a\#d\#(\mathbf{r} : \mathbf{e})C\ )$	[7]	
$\Rightarrow$	$(< p, AD\mathbf{E} >,$	$abc,$	$\#a\#d\#(r : e)C\ )$	[8]	konec

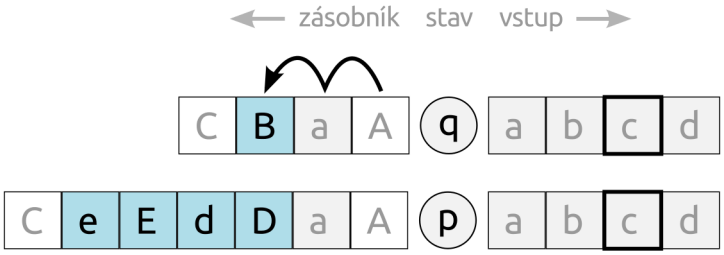
Kapitola 5

Zobecněný hluboký zásobníkový automat a jeho redukce

V předchozí kapitole jsem se věnovala automatu se zjednodušeným zápisem pravidel. Toho jsem dosáhla tak, že jsem vymezila několik povolených typů. Naskýtá se otázka, zda by nebylo výhodné zjednodušit přímo formu pravidla hlubokého zásobníkového automatu. Konkrétně by bylo možné vynechat hloubku expanze. Ta by tak nebyla definovaná v pravidle, ale specifikovaná v definici automatu. Tím bych dosáhla jistého zobecnění hlubokého zásobníkového automatu.

5.1 Zobecněný hluboký zásobníkový automat

Zobecněný hluboký zásobníkový automat (definice 5.1.1) vychází z hlubokého zásobníkového automatu, ale s hloubkou expanze pracuje jiným způsobem. Postupně prochází svým zásobníkem a aplikuje pravidlo typu $qA \to pv$ na nejvrchnější možný nevstupní symbol A. Dá se předpokládat, že tento model bude mít vyšší mocnost, neboť může expandovat symboly v libovolné hloubce.



Obrázek 5.1: Pravidlo $qB \to pDdEe$ lze ve zobecněném hlubokém zásobníkovém automatu v aktuální konfiguraci aplikovat právě tehdy, když neexistuje pravidlo s levou stranou qA.

Definice 5.1.1. *Zobecněný hluboký zásobníkový automat* je uspořádaná sedmice $M = (Q, \Sigma, \Gamma, R, s, S, F)$, kde Q, Σ, Γ, s, S, F jsou definované stejně jako u hlubokého zásobníkového automatu (definice 2.2.1) a R je konečná množina pravidel typu

$$(q, A, p, v) \in (Q \times (\Gamma - \Sigma) \times Q \times \Gamma^+), \text{ píšeme } qA \to pv.$$

Nechť X, Y jsou dvě konfigurace M. M provede expanzi z X do Y, pokud $X = (q, w, uAz)$, $Y = (p, w, uvz)$, $qA \to pv \in R$, kde p, $q \in Q$, $w \in \Sigma^*$, $A \in \Gamma$, u, v, $z \in \Gamma^*$, a platí:

$$\{X \mid qX \to p'v' \in R, \text{ kde } X \in \text{alph}(u) \cap (\Gamma - \Sigma),\ p' \in Q,\ v' \in \Gamma^+\} = \emptyset.$$

Definice 5.1.2. *Zobecněný hluboký zásobníkový automat s ε-pravidly* je zobecněný hluboký zásobníkový automat $M = (Q, \Sigma, \Gamma, R, s, S, F)$ rozšířený o vymazávací pravidla typu $qA \to p\varepsilon$, kde p, $q \in Q$, $A \in (\Gamma - \Sigma)$.

5.2 Ekvivalence se stavovými gramatikami

Když porovnám definici stavové gramatiky (2.3.4) s definicí zobecněného hlubokého zásobníkového automatu (5.1.1) je zřejmé, že se jedná o ekvivalentní modely.

V souvislosti s kapitolou 2.4 lze o tomto automatu prohlásit, že rodina jazyků, které přijímá, je identická s rodinou kontextových jazyků. Dále z definic 2.3.6 a 5.1.2 vyplývá, že zobecněný hluboký zásobníkový automat s ε-pravidly přijímá rodinu jazyků, která je identická s rodinou rekurzivně spočetných jazyků. Jinými slovy, tento automat má mocnost Turingova stroje.

5.3 Omezení počtu nevstupních symbolů

A. Meduna [2] dokázal, že každá stavová gramatika s ε-pravidly má svůj ekvivalentní protějšek se třemi nevstupními symboly. S ohledem na výsledek kapitoly 5.2 je na místě prozkoumat, zda lze téhož dosáhnout i u zobecněných hlubokých automatů.

V algoritmu 4 demonstruji princip redukce nevstupních symbolů na symboly 0, 1 a #. Hlavní myšlenka spočívá v reprezentaci všech zásobníkových symbolů pomocí nul a jedniček. Symbol # označuje fyzické dno zásobníku a umožňuje přesouvat symboly z vrcholu na jeho konec. Zároveň k symbolu # zavádím binární ekvivalent, který označuje logické dno zásobníku.

Princip simulace spočívá v rozpoznání symbolu na vrcholu zásobníku pomocí pravidel z množiny R_{find} a jeho přesunu na dno zásobníku, případně expanzi pravidlem z R_{exp}. Pokud dojde k expanzi, aplikují se pravidla z množiny R_{move}, která přesunou zbývající symboly na dno tak, aby se zásobník dostal do výchozího stavu pro další krok. Pokud pro žádný nevstupní symbol na zásobníku expanze neproběhne, automat nahradí binární reprezentace vstupních symbolů původními symboly pomocí pravidel z množiny R_{end}, smaže pomocné nevstupní symboly a přejde do koncového stavu.

Algoritmus 4. Převod zobecněného hlubokého zásobníkového automatu s ε-pravidly na ekvivalentní se třemi nevstupními symboly.

Vstup: $M = (Q, \Sigma, \Gamma, R, s, S, F)$
Výstup: $M_R = (Q_R, \Sigma_R, \Gamma_R, R_R, s_R, S_R, F_R)$

$\Sigma_R := \Sigma$
$\Gamma_R := \Sigma \cup \{0, 1, \#\}$
$R_R := R_{find} \cup R_{next} \cup R_{exp} \cup R_{move} \cup R_{end}$
$s_R :=< start >$
$S_R := \#$

$F_R := \{< end >\}$

Nechť $\Gamma = \{X_1, X_2, X_3, \ldots, X_n\}$ pro $n \geq 1$ je množina všech zásobníkových symbolů v M a X_0 značí symbol $\#$. Pak definujme funkci $\varphi : \Gamma \cup \{\#\} \to \{0,1\}^*$ tak, že $\varphi(X_i) = 0^i 1$ pro $i = 0, 1, 2, \ldots, n$.

Přidej do Q_R pro každé $q \in Q$, $0 \leq i \leq n$ stavy:
$< start >, < q, 0^i >, < q, 0^i 1 >, < q, 0^i >', < q, 0^i 1 >', < q, 0^i >'', < q, 0^i 1 >'', < end >$.

Pro každé $qA \to pB_0B_1 \ldots B_{j-1}B_j \in R$, kde $p, q \in Q$, $j \geq 0$, $B_0, B_1, \ldots, B_j \in \Gamma$ a $A \in (\Gamma - \Sigma)$, přidej $< q, \varphi(A) >$ do množiny Q_{exp} a do R_{exp} pravidla:

(i) $< start > \# \to < s, \varepsilon > \varphi(S)\varphi(\#)\#$,

(ii) $< q, \varphi(A) > \# \to < p, \varepsilon >' \varphi(B_0)\varphi(B_1)\varphi(B_2) \ldots \varphi(B_{j-1})\varphi(B_j)\#$.

Pro každé $q \in Q$, $0 \leq i < n$, $0 \leq j \leq n$ přidej do R_{find} pravidla:

(iii) $< q, 0^i > 0 \to < q, 0^{i+1} > \varepsilon$, $< q, 0^j > 1 \to < q, 0^j 1 > \varepsilon$,

(iv) $< q, 0^i >' 0 \to < q, 0^{i+1} >' \varepsilon$, $< q, 0^j >' 1 \to < q, 0^j 1 >' \varepsilon$,

(v) $< q, 0^i >'' 0 \to < q, 0^{i+1} >'' \varepsilon$, $< q, 0^j >'' 1 \to < q, 0^j 1 >'' \varepsilon$.

Přidej do R_{next} pravidla:

(vi) $< q, \varphi(X) > \# \to < q, \varepsilon > \varphi(X)\#$, kde $X \in \Gamma$, $q \in Q$, $< q, \varphi(X) > \notin Q_{exp}$,

(vii) $< q, \varphi(\#) > \# \to < q, \varepsilon >'' \varphi(\#)\#$, kde $q \in Q$.

Přidej do R_{move} pravidla:

(viii) $< q, \varphi(X) >' \# \to < q, \varepsilon >' \varphi(X)\#$, kde $X \in \Gamma$, $q \in Q$,

(ix) $< q, \varphi(\#) >' \# \to < q, \varepsilon > \varphi(\#)\#$, kde $q \in Q$.

Přidej do R_{end} pravidla:

(x) $< q, \varphi(X) >'' \# \to < q, \varepsilon >'' X\#$, kde $X \in \Sigma$, $q \in Q$,

(xi) $< q, \varphi(X) >'' \# \to < q, \varepsilon >'' \varphi(X)\#$, kde $X \in (\Gamma - \Sigma)$, $q \in Q$,

(xii) $< q, \varphi(\#) >'' \# \to < end > \epsilon$, kde $q \in Q$.

Z uvedeného algoritmu vyplývá, že redukce neovlivní mocnost hlubokého zásobníkového automatu s ε-pravidly. Pro automat bez ε-pravidel nelze tento postup použít. Zůstává tedy otázkou, zda je též zredukovatelný na tři nevstupní symboly.

Příklad 5.3.1. M je zobecněný hluboký zásobníkový automat a M_R je jeho zredukovaná varianta se třemi nevstupními symboly zkonstruovaná podle algoritmu 4. Nechť funkce φ je definovaná jako :

$$\varphi(x) = \begin{cases} 1 & \text{pro } x = \# \\ 01 & \text{pro } x = A \\ 001 & \text{pro } x = B \\ 0001 & \text{pro } x = C \\ 00001 & \text{pro } x = D \\ 000001 & \text{pro } x = d \end{cases}$$

Následující definice automatů jsou jen částečné z důvodu větší přehlednosti a pravidla jsou označena jednoznačným identifikátorem.

$$\begin{aligned} M = (\ & \{s, p, q, r, \ldots, f\}, \\ & \{a, b, c, d, e\}, \\ & \{a, b, c, d, e, A, B, C, D, E\}, \\ & \{qB \to pDd, \ldots\}, \\ & s, \\ & S, \\ & \{\text{f}\}\ ) \end{aligned}$$

$$\begin{aligned} M_R = (\ & \{\ < start >, < q, \varepsilon >, < q, 1 >, < q, 0 >, < q, 01 >, < q, 00 >, < q, 001 >, \\ & \quad < p, \varepsilon >, < p, 1 >, < p, 0 >, < p, 01 >, < p, 00 >, < p, 001 >, < p, 000 >, \\ & \quad < p, 0001 >, < p, \varepsilon >', < p, 1 >', < p, 0 >', < p, 01 >', < p, 00 >', \\ & \quad < p, 001 >', < p, 000 >', < p, 0001 >', \ldots, < end > \\ & \}\ \\ & \{\ a, b, c, d, e\ \}, \\ & \{\ a, b, c, d, e, 0, 1, \#\}, \\ & \{ \\ & \quad 1: \ \ < q, \varepsilon >\ 0\ \to < q, 0 >\ \varepsilon, \\ & \quad 2: \ \ < q, 0 >\ 1\ \to < q, 01 >\ \varepsilon, \\ & \quad 3: \ \ < q, 01 >\ \#\ \to < q, \varepsilon >\ 01\#, \\ & \quad 4: \ \ < q, \varepsilon >\ 0\ \to < q, 0 >\ \varepsilon, \\ & \quad 5: \ \ < q, 0 >\ 0\ \to < q, 00 >\ \varepsilon, \\ & \quad 6: \ \ < q, 00 >\ 0\ \to < q, 001 >\ \varepsilon, \\ & \quad 7: \ \ < q, 001 >\ \#\ \to < p, \varepsilon >'\ 00001000001\#, \\ & \quad 8: \ \ < p, \varepsilon >'\ 0\ \to < p, 0 >'\ \varepsilon, \\ & \quad 9: \ \ < p, 0 >'\ 0\ \to < p, 00 >'\ \ \varepsilon, \\ & \quad 10: \ < p, 00 >'\ 0\ \to < p, 000 >'\ \varepsilon, \\ & \quad 11: \ < p, 000 >'\ 1\ \to < p, 0001 >'\ \varepsilon, \\ & \quad 12: \ < p, 0001 >'\ \#\ \to < p, \varepsilon >'\ 0001\#, \\ & \quad 13: \ < p, \varepsilon >'\ 1\ \to < p, 1 >'\ \varepsilon, \\ & \quad 14: \ < p, 1 >'\ \#\ \to < p, \varepsilon >\ 1\#, \\ & \quad \ldots \\ & \}, \\ & < start >, \\ & \#, \\ & \{\ < end >\ \}\ ) \end{aligned}$$

Ukázka aplikace pravidla $qB \to pDd$ v konfiguraci (q, abc, ABC) v zobecněném hlubokém zásobníkovém automatu zkonstruovaném podle algoritmu 4.

	$(< q, \varepsilon >,$	$abc,$	0100100011#)		$ABC\#$	
$\Rightarrow$	$(< q, \mathbf{0} >,$	$abc,$	100100011#)	[1]		
$\Rightarrow$	$(< q, 0\mathbf{1} >,$	$abc,$	00100011#)	[2]	$BC\#$	přečteno A
$\Rightarrow$	$(< q, \varepsilon >,$	$abc,$	0010001**01**#)	[3]	$BC\#\mathbf{A}$	přesun A
$\Rightarrow$	$(< q, \mathbf{0} >,$	$abc,$	010001101#)	[4]		
$\Rightarrow$	$(< q, 0\mathbf{0} >,$	$abc,$	10001101#)	[5]		
$\Rightarrow$	$(< q, 00\mathbf{1} >,$	$abc,$	0001101#)	[6]	$C\#A$	přečteno B
$\Rightarrow$	$(< p, \varepsilon >',$	$abc,$	0001101**00001000001**#)	[7]	$C\#A\mathbf{Dd}$	expanze B
$\Rightarrow$	$(< p, \mathbf{0} >',$	$abc,$	00110100001000001#)	[8]		
$\Rightarrow$	$(< p, 0\mathbf{0} >',$	$abc,$	0110100001000001#)	[9]		
$\Rightarrow$	$(< p, 00\mathbf{0} >',$	$abc,$	110100001000001#)	[10]		
$\Rightarrow$	$(< p, 000\mathbf{1} >',$	$abc,$	10100001000001#)	[11]	$\#ADd$	přečteno C
$\Rightarrow$	$(< p, \varepsilon >',$	$abc,$	1010000100000**0001**#)	[12]	$\#ADd\mathbf{C}$	přesun C
$\Rightarrow$	$(< p, \mathbf{1} >',$	$abc,$	01000010000010001#)	[13]	$ADdC$	dno zásobníku
$\Rightarrow$	$(< p, \varepsilon >,$	$abc,$	010001000001000**1**#)	[14]	$ADdC\#$	konec

5.4 Omezení počtu stavů

Pro každou stavovou gramatiku s ε-pravidly dle A. Meduny [2] platí, že existuje ekvivalentní stavová gramatika s právě třemi stavy. V algoritmu 5 ukazuji, že totéž platí pro zobecněné hluboké zásobníkové automaty.

Pokud automat může být jen v jednom ze tří stavů, je zřejmé, že informace o stavu simulovaného automatu musí být součástí každého nevstupního symbolu. Musela jsem tedy najít způsob, jak přepsat stavy ve všech nevstupních symbolech na aktuální hodnotu.

Zvolila jsem princip „kruhového přepisování“. Pokud nad množinou stavů definuji uspořádání, pak mohu přepsat první stav na druhý, druhý na třetí, ..., předposlední na poslední a poslední na první. Po konečném počtu kroků jsem schopna získat libovolný stav bez ohledu na jeho pořadí. Počet těchto kroků určuje symbol čítače, jehož hodnota se po každé změně stavu ve všech nevstupních symbolech sníží o jedna.

Zobecněný hluboký zásobníkový automat expanduje nejvrchnější nevstupní symbol, pro který je to možné. Proto ke každému nevstupnímu symbolu s upravených stavem přidávám apostrof. Pro takový symbol v daném stavu neexistuje pravidlo a automat se přesune na další symbol až po konec zásobníku. Po snížení hodnoty čítače se ze symbolů na zásobníku odstraní apostrofy a v případě, že je čítač vynulovaný, se provede expanze. Pokud expanze způsobí změnu stavu, vygeneruje se nový čítač a celý postup se opakuje.

Funkčnost tohoto principu je závislá na rozdělení pravidel mezi tři stavy tak, aby automat v žádném případě nemohl vykonat nežádaný krok. V algoritmu 5 řeším tento problém následovně. Množina R_α obsahuje pravidla pro expanzi, změnu hodnoty čítače a nastavení příznaku expanze. Pravidla z R_β mění stav v nevstupních symbolech a symboly značí apostrofem. Množina pravidel R_γ odstraňuje apostrofy z nevstupních symbolů a podle příznaku na konci zásobníku automat přejde do stavu pro expanzi nebo pro aktualizaci stavů.

Algoritmus 5. Převod zobecněného hlubokého zásobníkového automatu s ε-pravidly na ekvivalentní s redukcí na tři stavy.

Vstup: $M = (Q, \Sigma, \Gamma, R, s, S, F)$

Výstup: $M_R = (Q_R, \Sigma_R, \Gamma_R, R_R, s_R, S_R, F_R)$

$\Sigma_R := \Sigma$
$\Gamma_R := \Sigma \cup \Gamma'_R$
$Q_R := \{s_\alpha, s_\beta, s_\gamma\}$
$R_R := R_\alpha \cup R_\beta \cup R_\gamma$
$s_R := s_\alpha$
$S_R :=< start >$
$F_R := \{s_\alpha\}$

Nechť $Q = \{s_0, s_1, s_2, \ldots, s_{|Q-1|}\}$, kde $s_0 = s$ je počáteční stav v M. Nechť $s_{|Q|} = s_0$.

Pro každé $q \in Q$, $X \in (\Gamma - \Sigma)$, $j \in \{1, 2, 3, \ldots, |Q|\}$ přidej do Γ'_R symboly :
$< start >, < j >, < q, X >, < q, X >', < q, \# >, < q, \# >_{set}, < q, \# >_{exp}$.

Pro každé $qA \to pb_0B_1b_1B_2b_2 \ldots b_{j-1}B_jb_j \in R$, kde $p, q \in Q$, $j \geq 0$, $b_0, b_1, \ldots, b_j \in \Sigma^*$ a $A, B_1, B_2, \ldots, B_j \in (\Gamma - \Sigma)$. Nechť $p = s_x$ a $q = s_y$, kde x, y jsou indexy stavů p, q, pak označme $k = (x - y) \ \% \ |Q|$. Přidej do R_α pravidla:

(i) $s_\alpha < start > \to s_\alpha < s_0, S >< s_0, \# >$,

(ii) $s_\alpha < q, A > \to s_\alpha b_0 < q, B_1 > b_1 < q, B_2 > b_2 \ldots b_{j-1} < q, B_j > b_j$ pokud $p = q$,

(iii) $s_\alpha < q, A > \to s_\beta < k > b_0 < q, B_1 > b_1 < q, B_2 > b_2 \ldots b_{j-1} < q, B_j > b_j$ pro $p \neq q$.

Přidej do R_β pravidla:

(iv) $s_\beta < s_i, X > \to s_\beta < s_{i+1}, X >'$ pro každé $s_i \in Q$ a každé $X \in (\Gamma - \Sigma)$,

(v) $s_\beta < s_i, \# > \to s_\alpha < s_{i+1}, \# >_{set}$ pro každé $s_i \in Q$.

Přidej do R_α pravidla:

(vi) $s_\alpha < j > \to s_\gamma < j - 1 >$ pro každé $j \in \{2, 3, \ldots, |Q|\}$,

(vii) $s_\alpha < 1 > \to s_\alpha \varepsilon$,

(viii) $s_\alpha < s_i, \# >_{set} \to s_\gamma < s_i, \# >_{exp}$ pro každé $s_i \in Q$,

(ix) $s_\alpha < q, \# > \to s_\alpha \varepsilon$ pro každé $q \in F$.

Přidej do množiny R_γ pravidla:

(x) $s_\gamma < s_i, X >' \to s_\gamma < s_i, X >$ pro každé $s_i \in Q$ a každé $X \in (\Gamma - \Sigma)$,

(xi) $s_\gamma < s_i, \# >_{set} \to s_\beta < s_i, \# >$ pro každé $s_i \in Q$,

(xii) $s_\gamma < s_i, \# >_{exp} \to s_\alpha < s_i, \# >$ pro každé $s_i \in Q$.

Každý zobecněný hluboký zásobníkový automat s ε-pravidly lze převést na ekvivalentní se třemi stavy, jak vyplývá z algoritmu 5. Otázka, zda lze stejně zredukovat automat bez ε-pravidel, zůstává otevřená.

Příklad 5.4.1. M je zobecněný hluboký zásobníkový automat a M_R je jeho zredukovaná varianta zkonstruovaná podle algoritmu 5. Definice automatů jsou pro větší přehlednost neúplné. Na pravidla se odkazuji pomocí identifikátorů.

$$
\begin{aligned}
M = (\ & \{s, p, q, r, \ldots, f\}, \\
& \{a, b, c, d, e\}, \\
& \{a, b, c, d, e, A, B, C, D, E\}, \\
& \{pB \to rDd, \ldots\}, \\
& s, \\
& S, \\
& \{\mathrm{f}\}\ )
\end{aligned}
$$

$$
\begin{aligned}
M_R = (\ & \{\ s_\alpha, s_\beta, s_\gamma\ \}, \\
& \{\ a, b, c, d, e\ \}, \\
& \{\ a, b, c, d, e, < start >, < 1 >, < 2 >, \ldots, < p, A >, < p, B >, < p, C >, \\
& \quad < p, D >, < p, A >', < p, B >', < p, C >', < p, D >', < q, A >, < q, B >, \\
& \quad < q, C >, < q, D >, < q, A >', < q, B >', < q, C >', < q, D >', < r, A >, \\
& \quad < r, B >, < r, C >, < r, D >, < r, A >', < r, B >', < r, C >', < r, D >', \\
& \quad < p, \# >, < p, \# >_{set}, < p, \# >_{exp}, < q, \# >, < q, \# >_{set}, < q, \# >_{exp}, \\
& \quad < r, \# >, < r, \# >_{set}, < r, \# >_{exp}, \ldots \\
& \}, \\
& \{ \\
& \quad 1: \ \ s_\alpha\ < p, B > \to\ s_\beta < 2 > < p, D >\ d, \\
& \quad 2: \ \ s_\beta\ < q, A > \to\ s_\beta\ < q, A >', \\
& \quad 3: \ \ s_\beta\ < q, D > \to\ s_\beta\ < q, D >', \\
& \quad 4: \ \ s_\beta\ < p, \# > \to\ s_\alpha\ < q, \# >_{set}, \\
& \quad 5: \ \ s_\alpha\ < 2 > \to\ s_\gamma\ < 1 >, \\
& \quad 6: \ \ s_\gamma\ < q, A >' \to\ s_\gamma\ < q, A >, \\
& \quad 7: \ \ s_\gamma\ < q, D >' \to\ s_\gamma\ < q, D >, \\
& \quad 8: \ \ s_\gamma\ < q, \# >_{set} \to\ s_\beta\ < q, \# >, \\
& \quad 9: \ \ s_\beta\ < r, A > \to\ s_\beta\ < r, A >', \\
& \quad 10: \ s_\beta\ < r, D > \to\ s_\beta\ < r, D >', \\
& \quad 11: \ s_\beta\ < r, \# > \to\ s_\alpha\ < r, \# >_{set}, \\
& \quad 12: \ s_\alpha\ < 1 > \to\ s_\alpha\ \varepsilon, \\
& \quad 13: \ s_\alpha\ < r, \# >_{set} \to\ s_\gamma\ < r, \# >_{exp}, \\
& \quad 14: \ s_\gamma\ < r, A >' \to\ s_\gamma\ < r, A >, \\
& \quad 15: \ s_\gamma\ < r, D >' \to\ s_\gamma\ < r, D >, \\
& \quad 16: \ s_\gamma\ < r, \# >_{exp} \to\ s_\alpha\ < r, \# >, \\
& \quad \ldots \\
& \}, \\
& s_\alpha, \\
& < start >, \\
& \{\ s_\alpha\ \}\ )
\end{aligned}
$$

Ukázka simulace pravidla $pB \to rDd$ v konfiguraci (p, abc, ABc) v zobecněném hlubokém zásobníkovém automatu M_R zkonstruovaném z automatu M podle algoritmu 5.

	$(s_\alpha,\ abc,\ < p, A >< p, B > c < p, \# >)$		
$\Rightarrow$	$(\mathbf{s}_\beta,\ abc,\ < p, A > < \mathbf{2} >< \mathbf{p}, \mathbf{D} > \mathbf{d}c < p, \# >)$	[1]	expanze $pB \to rDd$
$\Rightarrow$	$(s_\beta,\ abc,\ < \mathbf{q}, \mathbf{A} >' < 2 >< p, D > dc < p, \# >)$	[2]	přepis stavů q na p
$\Rightarrow$	$(s_\beta,\ abc,\ < q, A >' < 2 > < \mathbf{q}, \mathbf{D} >' dc < p, \# >)$	[3]	
$\Rightarrow$	$(\mathbf{s}_\alpha,\ abc,\ < q, A >' < 2 >< q, D >' dc < \mathbf{q}, \# >_{\mathbf{set}})$	[4]	nastavení příznaku *set*
$\Rightarrow$	$(\mathbf{s}_\gamma,\ abc,\ < q, A >' < \mathbf{1} > < q, D >' dc < q, \# >_{set})$	[5]	snížení hodnoty čítače
$\Rightarrow$	$(s_\gamma,\ abc,\ < \mathbf{q}, \mathbf{A} > < 1 >< q, D >' dc < q, \# >_{set})$	[6]	odznačení symbolů
$\Rightarrow$	$(s_\gamma,\ abc,\ < q, A >< 1 > < \mathbf{q}, \mathbf{D} > dc < q, \# >_{set})$	[7]	
$\Rightarrow$	$(\mathbf{s}_\beta,\ abc,\ < q, A >< 1 >< q, D > dc < \mathbf{q}, \# >)$	[8]	odstranění příznaku *set*
$\Rightarrow$	$(s_\beta,\ abc,\ < \mathbf{r}, \mathbf{A} >' < 1 >< q, D > dc < q, \# >)$	[9]	přepis stavů p na r
$\Rightarrow$	$(s_\beta,\ abc,\ < r, A >' < 1 > < \mathbf{r}, \mathbf{D} >' dc < q, \# >)$	[10]	
$\Rightarrow$	$(\mathbf{s}_\alpha,\ abc,\ < r, A >' < 1 >< r, D >' dc < \mathbf{r}, \# >_{\mathbf{set}})$	[11]	nastavení příznaku *set*
$\Rightarrow$	$(s_\alpha,\ abc,\ < r, A >' < r, D >' dc < r, \# >_{set})$	[12]	vynulování čítače
$\Rightarrow$	$(\mathbf{s}_\gamma,\ abc,\ < r, A >' < r, D >' dc < \mathbf{r}, \# >_{\mathbf{exp}})$	[13]	nastavení příznaku *exp*
$\Rightarrow$	$(s_\gamma,\ abc,\ < \mathbf{r}, \mathbf{A} > < r, D >' dc < r, \# >_{exp})$	[14]	odznačení symbolů
$\Rightarrow$	$(s_\gamma,\ abc,\ < r, A > < \mathbf{r}, \mathbf{D} > dc < r, \# >_{exp})$	[15]	
$\Rightarrow$	$(\mathbf{s}_\alpha,\ abc,\ < r, A >< r, D > dc < \mathbf{r}, \# >)$	[16]	konec

Kapitola 6

Aplikace gdeep_pda

V této kapitole popisuji návrh a implementaci konzolové aplikace pro redukci zobecněných hlubokých zásobníkových automatů a syntaktickou analýzu řetězců, které tyto automaty přijímají. Na aplikaci demonstruji implementaci algoritmů 4 a 5 navržených v kapitole 5.

6.1 Návrh aplikace a formátu vstupního souboru

Při návrhu aplikace jsem vycházela z kapitoly 5 věnované zobecněným hlubokým zásobníkovým automatům. Dále bylo třeba navrhnout základní komponenty aplikace, postup pro syntaktickou analýzu řetězce a formát zápisu automatu.

Rozvržení aplikace

Aplikace se skládá z několika částí. Práci se zásobníkovým automatem umožňují moduly `automaton`, `parser`, `symbol_reduction` a `state_reduction`. Modul `automaton` obsahuje třídu reprezentující zobecněný hluboký zásobníkový automat, `parser` zprostředkovává načtení zápisu automatu z řetězce a `symbol_reduction` spolu se `state_reduction` slouží k redukci symbolů nebo stavů automatu. Činnost aplikace pak zajišťují moduly `application`, `error` a `library`, kde `application` umožňuje běh aplikace, `error` definuje chybové stavy a `library` slouží jako knihovna funkcí.

Syntaktická analýza řetězce

Zobecněné hluboké zásobníkové automaty jsou nedeterministické, což je třeba u syntaktické analýzy zohlednit. Vzhledem k demonstračním účelům aplikace jsem se rozhodla použít metodu shora dolů s návratem, která je z hlediska časové složitosti neefektivní, ale jednoduchá na implementaci.

Automat postupně provádí jednotlivé kroky derivace tak, že na nejvrchnější možný nevstupní symbol aplikuje první nalezené pravidlo. Pokud je na vrcholu zásobníku vstupní symbol, provede se pop operace. Jestliže tuto operaci nelze provést, automat se vrátí do předchozí konfigurace a pokusí se aplikovat jiné pravidlo. V případě neúspěchu je opět proveden návrat. Pokud se automat vrátí za výchozí konfiguraci, pak analyzovaný řetězec není řetězcem jazyka přijímaného tímto automatem. V opačném případě algoritmus skončí, jakmile je přečten celý vstupní řetězec, automat je v konečném stavu a zásobník je prázdný.

Automat prochází derivační strom do hloubky. Je proto vhodné omezit hloubku stromu, jinak by analýza řetězce mohla vést k zacyklení aplikace. Pro ukládání konfigurací a obno-

vení konfigurace předcházející lze použít zásobník. Po každém kroku automatu se na tento zásobník vloží aktuální konfigurace a pořadové číslo aplikovaného pravidla. Při neúspěchu se automat nastaví podle konfigurace na vrcholu zásobníku. Po úspěšné analýze lze pomocí tohoto zásobníku zrekonstruovat jednotlivé kroky derivace.

Formát zápisu automatu

Zápis zobecněného zásobníkového automatu vychází z jeho definice. Hlavním kritériem formátování bylo, aby výstupem programu byl automat, který se může použít jako vstup. Zároveň v souvislosti s redukcí jsem musela umožnit vytvářet složitější zápisy stavů a nevstupních symbolů, aby se daly použít k uchovávání nezbytných informací. Příklad formátování je k dispozici v 6.1.1. Zevrubnější popis je součástí manuálu k aplikaci.

Příklad 6.1.1. Ukázka zápisu zobecněného hlubokého automatu pro aplikaci `gdeep_pda`.

```
// Hluboký zásobníkový automat:
(
    // množina stavů
    { <s1>, <s2>, <s3>, <s4>, <s5> },
    // množina vstupních symbolů
    { a, b, c },
    // množina zásobníkových symbolů
    { A, B, C, a, b, c },
    // množina pravidel
    {
        <s1> A -> <s2> a A b B c C,
        <s2> A -> <s3> a A,
        <s3> B -> <s4> b B,
        <s4> C -> <s2> c C,
        <s2> A -> <s5>,            // epsilon pravidlo
        <s5> B -> <s5> '',         // epsilon pravidlo
        <s5> C -> <s5> '' '' ''    // epsilon pravidlo
    },
    <s1>,     // počáteční stav
    A,        // počáteční symbol
    { <s5> }  // množina koncových stavů
)
```

6.2 Implementace aplikace

Aplikaci `gdeep_pda` jsem implementovala v jazyce Python verze 3.2 jako spustitelný balík. Cílovou platformou byly operační systémy unixového typu. Na jiných platformách aplikace nebyla testována.

Činnost aplikace

Po spuštění je zavolána funkce `main()` z modulu `application`. Ta zpracuje parametry, načte vstup do řetězce a řetězec zpracuje pomocí parseru typu `GDPParser`. Vstupní automat se pak formátovaně vypíše na výstup, nebo se získá jeho zredukovaná varianta pomocí

objektu třídy `StateReduction`, případně `SymbolReduction`, a vypíše se zredukovaný automat. Modul `library` poskytuje funkce pro výpis nápovědy, zpracování parametrů a práci se vstupy a výstupy.

Model zobecněného hlubokého zásobníkového automatu

Třída `GDP` z modulu `automaton` reprezentuje zobecněný hluboký zásobníkový automat. Komponenty automatu jsou implementované jako instanční proměnné `Q`, `Sigma`, `Gamma`, `R`, `s`, `S` a `F`, jejichž pojmenování odpovídá definici 5.1.1.

Proměnné `Q`, `Sigma`, `Gamma`, `F` jsou množiny řetězců, `s`, `S` jsou řetězce a `R` je množina uspořádaných čtveřic `(q, A, p, v)`, kde `q`, `A`, `p` jsou řetězce, `v` je n-tice řetězců a čtveřice je reprezentací pravidla $qA \to pv$. K snazšímu sestavování komplikovanějších pravidel slouží třída `GDP_rule`, kde `q`, `A`, `p`, `v` jsou proměnné instance a metoda `get()` vrací uspořádanou čtveřici.

Metoda `validate()` kontroluje sémantickou správnost automatu a v případě chyby vyvolá výjimku `EPDA`. Metoda `serialize()` vrací řetězec s formátovaným zápisem automatu.

Parsování automatu z řetězce

Původně jsem vstupní řetězec zpracovávala výhradně pomocí regulárních výrazů, ale pro obsáhlejší automaty tento postup výrazně zpomaloval aplikaci. Dále jsem zvažovala použít konečný automat pro lexikální analýzu a syntaxi analyzovat metodou shora dolů, ale takové řešení by bylo velmi robustní a neumožňovalo jednoduše měnit formát zápisu. Nakonec jsem oba postupy zkombinovala. Stavy a symboly zpracovávám regulárními výrazy, automat a pravidla konečným automatem. Konečný automat je pak řízen pravidly, která jsou specifikovaná formou zjednodušených regulárních výrazů. Činnost parseru je tak nezávislá na formátu zápisu automatu a bylo by snadné umožnit analýzu dalších typů automatů v různých formátech.

Zpracování vstupu zajišťuje třída `GDPParser` z modulu `parser`. Při inicializaci se nastaví vzory (regulární výrazy a pravidla) a analýza řetězce se spustí zavoláním metody `run()`. Parsování zajišťují funkce `match()`, `matchStr()`, `matchItem()`, `matchList()` a `matchGroup()`, které vrací dvojici prvek, index. Prvek je nalezený řetězec nebo seznam prvků, index označuje pozici v řetězci pro další parsování. Metoda `match()` pak slouží k aplikaci regulárního výrazu, `matchStr()` porovnává řetězec, `matchItem()` zavolá podle typu vzoru odpovídající metodu, `matchList()` volá matchItem() pro všechny vzory v předaném seznamu a `matchGroup()` cyklicky volá `matchItem()` pro analýzu položek prokládaných oddělovačem. Nakonec metoda `run()` vrátí načtený automat typu `GDP`, nebo vyvolá výjimku `EPDA`.

Redukce nevstupních symbolů

Třída `SymbolReduction` z modulu `symbol_reduction` umožňuje provádět redukci počtu nevstupních symbolů nad zobecněnými hlubokými zásobníkovými automaty. Metoda `run()` obdrží automat typu `GDP` a vrátí automat zkonstruovaný podle algoritmu 4 z kapitoly 5. Metoda `getCodingFunction()` vrátí kódovací funkci pro symboly. Tato funkce je reprezentovaná datovým typem slovník, kde vstupem funkce je klíč a výstupem funkce hodnota. Následně jsou metodami `construct_Q()` a `construct_R()` sestaveny množiny stavů a pravidel a je zkonstruován nový automat. Jeho správnost se ověřuje metodou `validate()` z třídy `GDP`.

Redukce stavů

K redukci počtu stavů slouží třída `StateReduction` z modulu `state_reduction`. Konstrukci automatu se třemi stavy provádí metoda `run()` podle algoritmu 5 z kapitoly 5. Metoda `getIndexFunction()` seřadí stavy automatu podle abecedy a vrátí funkci pro určení pozice stavu v tomto uspořádání. Metoda `getStateFunction()` pak definuje a vrátí funkci, která pro každý stav automatu vrací stav následující a pro poslední stav stav první. Poté metoda `construct_Gamma()` zkonstruuje množinu zásobníkových symbolů a metoda `construct_R()` sestaví množinu pravidel. Nakonec je vytvořen a inicializován výsledný automat a ověřena jeho správnost.

Analýza vstupního řetězce

Syntaktickou analýzu řetězce umožňuje třída `GDP` z modulu `automaton`. Metoda `analyze()` hledá derivaci řetězce pomocí postupu navrhovaném v kapitole 6.1. Aktuální konfiguraci automatu lze získat voláním `getConfiguration()`, `saveDerivation()` a `loadDerivation()` umožňují ukládat konfigurace na zásobník a obnovit stav automatu podle poslední uložené konfigurace. Pokud analýza proběhne úspěšně, pak funkce `tableprint()` z modulu `library` převede kroky derivace do řetězce a výsledek se vypíše na výstup. Hloubku derivace lze nastavit parametrem pro maximální počet kroků.

Ošetření chybových stavů

V modulu `error` jsou definované výjimky `Error`, `EPARAM`, `EIO` a `EPDA`. `Error` je základní třída, kterou dědí ostatní výjimky. `EPARAM` indikuje chybu v parametrech, `EIO` chybu při práci se soubory a `EPDA` označuje chybu v zásobníkovém automatu. Pokud je vyvolána výjimka, na standardní chybový výstup se vypíše chybová hláška a aplikace skončí s odpovídající návratovou hodnotou.

6.3 Testování

Pro účely testování jsem použila framework `unittest` a vytvořila několik testovacích případů k otestování hlavních komponent aplikace: zpracování parametrů, zpracování vstupu, analýzu řetězce a redukci automatu. Všechny testy jsou součástí balíku `test` a spustitelné příkazem `python3 test.py`.

6.4 Spuštění aplikace

Aplikace `gdeep_pda` pro svůj běh vyžaduje Python verze 3.2. Manuál s podrobnějšími informacemi je k dispozici v souboru `MANUAL`. Aplikaci lze spustit příkazem:

```
python3 gdeep_pda.py [-h|--help] [--input=filename] [--output=filename]
                     [--reduce-states] [--reduce-symbols]
                     [--analyze-string="s t r i n g"] [--max-steps=n]
```

Kapitola 7

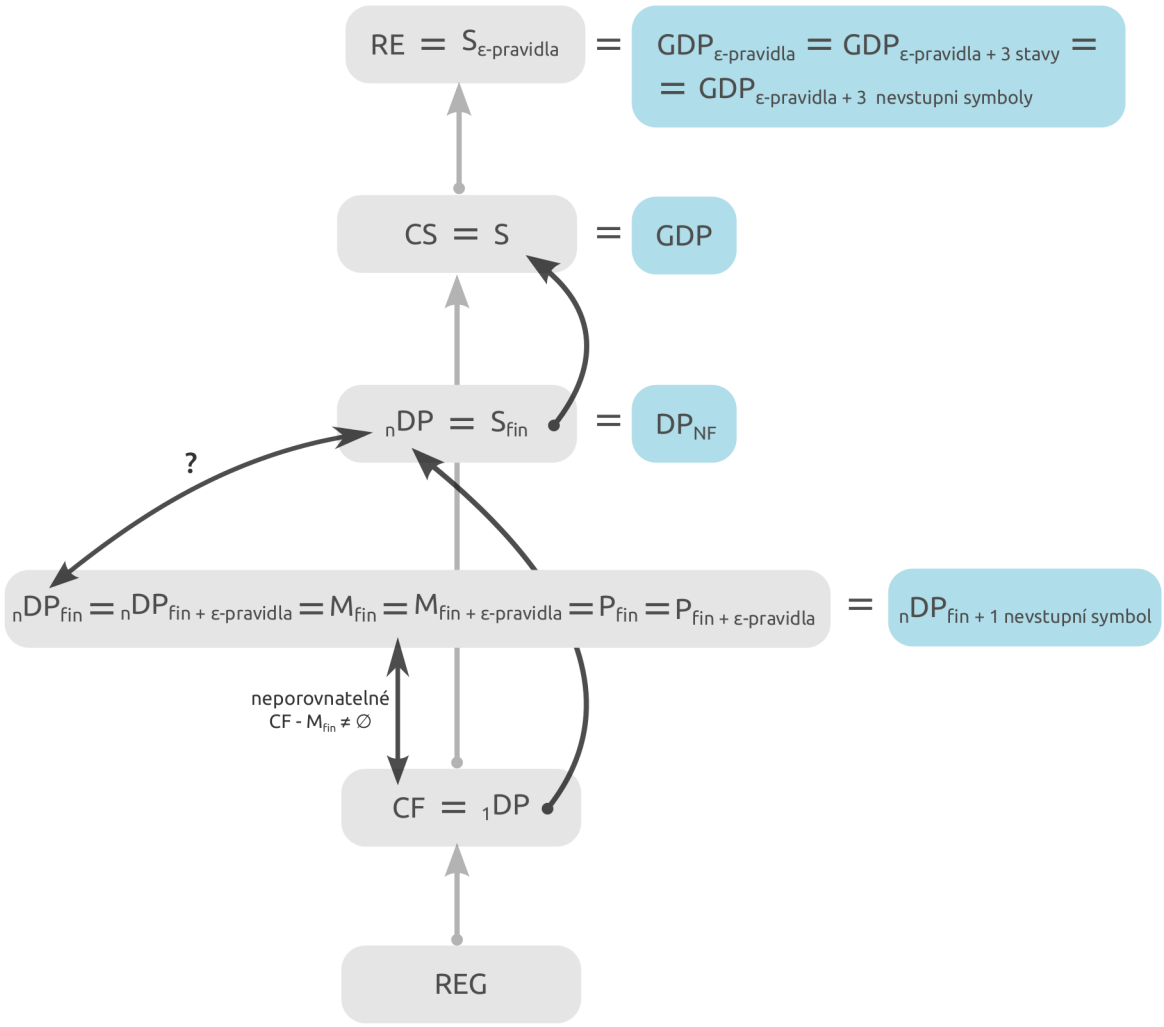
Závěr

V této práci jsem se zabývala hlubokými zásobníkovými automaty z pohledu počtu nevstupních symbolů a stavů. Cílem bylo zavést nové modifikace tohoto typu automatu a dosáhnout tak jeho částečného zjednodušení. Po dohodě s vedoucím práce jsem původní zadání rozšířila o problematiku normální formy a zobecněné varianty hlubokých zásobníkových automatů.

Dokázala jsem, že hluboký zásobníkový automat konečného indexu s jedním nevstupním symbolem je ekvivalentní s programovými gramatikami konečného indexu. Znamená to, že omezením počtu nevstupních symbolů na jeden se jeho mocnost nezměnila. Dále jsem zavedla normální formu hlubokého zásobníkového automatu se čtyřmi typy pravidel a docílila tak zjednodušení jejich zápisu. Nakonec jsem definovala zobecněný hluboký zásobníkový automat. Toto zobecnění spočívá v odstranění hloubky expanze z pravidel. Místo toho je upraveno chování automatu tak, aby expandoval nejvrchnější možný nevstupní symbol na svém zásobníku. Tento typ automatu s povolenými ε-pravidly má mocnost Turingova stroje a platí, že zredukováním počtu nevstupních symbolů, případně stavů, na tři se síla automatu nezmění. Srovnání mocností zavedených automatů a zasazení do širšího kontextu je k dispozici na diagramu 7.1.

Algoritmy pro redukci zobecněného hlubokého zásobníkového automatu jsem po dohodě s vedoucím práce implementovala v konzolové aplikaci `gdeep_pda`. Ta umožňuje převádět vstupní automat na zredukovaný, případně provést syntaktickou analýzu vstupního řetězce, a výsledek vypsat na výstup. Aplikace je implementovaná v jazyce Python a slouží k demonstračním účelům.

Uvedené algoritmy popisují, jakým způsobem je možné jeden model převést na druhý, z čehož lze vyvozovat závěry o jejich ekvivalenci. Bylo by však vhodné ověřit správnost algoritmů formálním způsobem. Z navržených redukcí lze dále odvodit, že omezení počtu prvků jedné komponenty automatu se projevuje výrazným zvýšením počtu prvků v ostatních komponentách. Konkrétně, když jsem omezila počet nevstupních symbolů, zvýšila se kardinalita množin pravidel a stavů a po omezení počtu stavů, narostla kardinalita množin pravidel a nevstupních symbolů. Je tedy pravděpodobné, že není možné současně omezit počet stavů a počet nevstupních symbolů se zachováním mocnosti automatu. Jedná se však o zajímavou problematiku vhodnou k dalšímu prostudování.



Obrázek 7.1: Zařazení automatů do kontextu jazykových rodin. Význam jednotlivých zkratek je vysvětlen v kapitole 2.4. Dále $\mathbf{DP}_{NF}$ označuje rodinu jazyků přijímaných hlubokými zásobníkovými automaty v normální formě a **GDP** rodinu jazyků přijímaných zobecněnými hlubokými zásobníkovými automaty.

Literatura

[1] Dassow, J.; Păun, G.: *Regulated Rewriting in Formal Language Theory*. Berlin: Akademie–Verlag, 1989, ISBN 3-05-500408-6, 308 s.

[2] Horváth, G.; Meduna, A.: On State Grammars. *Acta Cybernetica*, ročník 1988, č. 8, 1988: s. 237–245, ISSN 0324-721X.

[3] Kasai, T.: An Hierarchy Between Context-Free and Context-Sensitive Languages. *Journal of Computer and System Sciences*, ročník 4, č. 5, 1970: s. 492–508, ISSN 0022-0000.

[4] Křivka, Z.; Meduna, A.; Schönecker, R.: Generation of Languages by Rewriting Systems that Resemble Automata. *International Journal of Foundations of Computer Science*, ročník 17, č. 5, 2006: s. 1223–1229, ISSN 0129-0541.

[5] Meduna, A.: *Automata and Languages: Theory and Applications*. London: Springer, 2005, ISBN 81-8128-333-3, 916 s.

[6] Meduna, A.: Deep Pushdown Automata. *Acta Informatica*, ročník 2006, č. 98, 2006: s. 114–124, ISSN 0001-5903.

[7] Meduna, A.: Finitely Expandable Deep PDAs. In *Automata, Formal Languages and Algebraic Systems*, Hong Kong: Hong Kong University of Science and Technology, 2010, ISBN 981-4317-60-8, s. 113–123.

Příloha A

Obsah CD

Přiložené CD obsahuje:

- písemnou zprávu ve formátu PDF,
- zdrojové texty písemné zprávy,
- zdrojové texty programu a manuál,
- testovací skript,
- ukázkové vstupní soubory.