VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

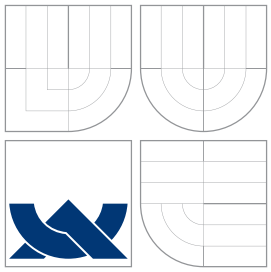
AUTOMATY S OHRANIČENÝM POČTEM ČISTÝCH ZÁSOBNÍKŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

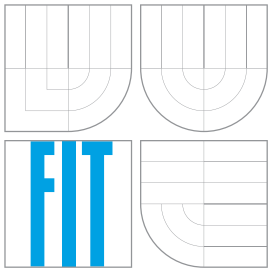
AUTOR PRÁCE
AUTHOR

ONDŘEJ SOUKUP

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

AUTOMATY S OHRANIČENÝM POČTEM ČISTÝCH ZÁSOBNÍKŮ

AUTOMATA WITH A LIMITED NUMBER OF PURE PUSHDOWNS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ONDŘEJ SOUKUP
AUTHOR

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

BRNO 2011

Abstrakt

Zásobníkové automaty s omezeným počtem čistých zásobníků jsou specializací čistých zásobníkových automatů. Jejich síla je zkoumána především z pohledu kombinatoriky na slovech. V jazyku definovaném automatem je zaveden termín závislost, který označuje vztah mezi částmi řetězců jazyka. Je ukázáno, jak postupuje definovaný automat při zpracování závislostí. Dále je odvozen vztah mezi rozložením závislostí v jazyku a nutným počtem zásobníků automatu. Je definována nekonečná hierarchie jazyků korespondujících k automatům v závislosti na počtu jejich zásobníků. Nakonec je zkoumáno zařazení třídy jazyků definované zásobníkovými automaty s omezeným počtem čistých zásobníků do Chomského hierarchie jazyků, ovšem je zjištěno, že s Chomského hierarchií nekorespondují. Na základě výsledku je jako další vývoj navrhováno zkoumání možných úprav modelů. Pro demonstraci vlastností zkoumaných automatů je následně vytvořen simulační program.

Abstract

Pushdown automata with limited number of pure pushdowns are specialization of pure pushdown automata. Examination of their power is mainly focused on the view of combinatorics of words. In language defined by automata we introduce term of dependence, which denotes relation between parts of strings in language. It is shown, how the defined automata proceed on processing of dependencies. Then the relation between distribution of dependencies in language and required number of pushdowns is derived. Also is defined the infinite hierarchy of languages corresponding to automata in dependence on number of their pushdowns. At the end belonging of the class of languages defined by pushdown automata with limited number of pure pushdowns to Chomsky hierarchy of languages is studied, but it is discovered, that they do not correspond to Chomsky hierarchy. Based on this result it is proposed to study possible changes of models as a future development. Then the simulating program is created to illustrate features of examined automata.

Klíčová slova

Zásobníkové automaty, čisté zásobníky, kombinatorika na slovech

Keywords

Pushdown automata, pure pushdowns, combinatorics of words

Citace

Ondřej Soukup: Automaty s ohraničeným počtem čistých zásobníků, bakalářská práce, Brno, FIT VUT v Brně, 2011

Automaty s ohraničeným počtem čistých zásobníků

Prohlášení

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

.....
Ondřej Soukup
15. května 2011

Poděkování

Předem práce bych rád poděkoval prof. Alexandru Medunovi za odborné vedení a především za mé přivedení do oblasti formálních jazyků a teorie automatů. Díky zájmu, který ve mne vytrvalá iniciativa vzbudila mohla tato práce vzniknout.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Tato práce se zabývá *zásobníkovými automaty s omezeným počtem čistých zásobníků*. Jedná se o specializaci čistých zásobníkových automatů vzniklou jejich rozšířením z jediného zásobníku na n. Rozšíření zvýší jejich sílu nad rámec původních modelů, do jaké míry se výpočetní síla zvýší budeme zkoumat. Na problematiku se zaměříme z pohledu *kombinatoriky na slovech* (viz [2]). Budeme se zabývat morfologickými vlastnostmi řetězců zpracovávaného jazyka. Výchozím zdrojem pro práci je především [4] (případně [3]), přesto si ale zavedeme automaty z části odlišným způsobem. V kapitole 2 uvedeme nejdříve několik základních definic, na které budeme moci navázat zavedením zkoumaných automatů. Abychom mohli korektně zavést zásobníkové automaty s omezeným počtem čistých zásobníků, je třeba nejdříve zavést pojem čistých zásobníkových automatů. Ty jsou specializací zásobníkových automatů. Zásobníkové automaty rozšiřují možnosti konečných automatů. V kapitole 3 pak zásobníkové automaty s omezeným počtem čistých zásobníků zavedeme, stanovíme jejich vlastnosti a omezení a v následujícím textu budeme zjišťovat jejich vliv na sílu automatů. Budeme se snažit co nejvíce zúžit zkoumanou problematiku, aby vynikly vlastnosti, na které se zaměříme. Můžeme zúžit vstupní abecedu, při zkoumání změn na zásobnících například zanedbat vliv změn stavů automatu atd. Dále zavedeme pojem *závislosti* v přijímaném jazyku. Tím budeme rozumět vztah částí řetězců jazyka, například shodu délek dvou podřetězců každého řetězce jazyka. Přestože závislostí můžeme obecně myslet velmi složitý vztah, omezíme se pouze na jednoduché závislosti, protože podstatou bude zkoumání vlivu jejich rozložení ve vstupním řetězci na práci automatu. Identifikujeme několik základních možností vzájemného rozložení dvou závislostí a následně zobecníme problém na vzájemné rozložení více závislostí v řetězci. Zjistíme, jak definované automaty zpracovávají závislost a jak se jejich činnost změní, pokud je třeba zpracovávat více závislostí současně. Prostudujeme rozdíly ve zpracování vstupního řetězce v návaznosti na dříve definované možnosti vzájemného rozložení závislostí v řetězci. Zaměříme se hlavně na souvislost potřebného počtu zásobníků nutných ke zpracování řetězců jazyka a rozložení závislostí v řetězcích jazyka. Nakonec se budeme zabývat korespondencí jazyků, které zavedené automaty definují, k Chomského hierarchii jazyků. Na základě výsledku pak stanovíme další možný postup výzkumu v kapitole 4. Abychom mohli demonstrovat zjištěné vlastnosti a sílu zkoumaných automatů, bude vytvořena aplikace simulující jejich činnost při zpracování vstupního řetězce na modelu vytvořeném podle zadané definice. Popis použití a ovládání aplikace lze nalézt v příloze B.

Kapitola 2

Úvodní definice

Dříve, než se začneme věnovat ústřednímu tématu této práce, je třeba definovat základní pojmy, které budeme používat. Předpokládáme, že čtenář je seznámen s teorií množin a základy teorie formálních jazyků, především s Chomského hierarchií jazyků. Nebudeme provádět hlubší zkoumání pojmů definovaných v této kapitole ani důkazy uvedených tvrzení. Zájemce o bližší studium problematiky může nahlédnout například do [8].

2.1 Abecedy, řetězce, jazyky

Definice 2.1.1. *Abeceda* je konečná neprázdná množina. Prvky abecedy budeme nazývat symboly.

Příkladem abecedy může být $V = \{a, b, c\}$, potom a, b, c jsou symboly abecedy V.

Definice 2.1.2. Mějme abecedu V. Konečnou posloupnost symbolů z V nazveme *řetězec* nad abecedou V. Prázdnou posloupnost nazveme *prázdný řetězec* a budeme značit ε.

Obecně je možné uvažovat i nekonečné řetězce, v této práci se jimi ale zabývat nebudeme. Mějme řetězce x, y, z nad abecedou V. Pak jejich zřetězení $w = xyz$ je také řetězcem nad V, přičemž x je *prefix*, y *sufix* a x, y, z jsou *podřetězce* řetězce w. Buď $n \in \mathbb{N}$. *n-tou mocninou* řetězce w budeme rozumět n-násobné zřetězení řetězce w. Zřetězení budeme častěji nazývat *konkatenací*, mocninu *iterací*.

Definice 2.1.3. Mějme abecedu V a řetězec $w = v_1 v_2 \dots v_n$, $v_i \in V$. *Délkou řetězce* w rozumíme $|w| = n$. Délka prázdného řetězce je nula $|\varepsilon| = 0$.

Mějme abecedu V. Jako V^* budeme značit množinu všech řetězců nad abecedou V. V^+ pak bude značit množinu všech neprázdných řetězců nad V. Platí $V^+ = V^* - \{\varepsilon\}$.

Definice 2.1.4. Mějme abecedu V. Pro každé L, $L \subseteq V^*$, platí, L je *jazyk* nad V.

Pro jazyky platí množinové operace stejně jako operace řetězcové s rozdílem, že jsou aplikovány postupně na všechny možné kombinace řetězců a výsledkem je sjednocení dílčích výsledků.

2.2 Konečné automaty

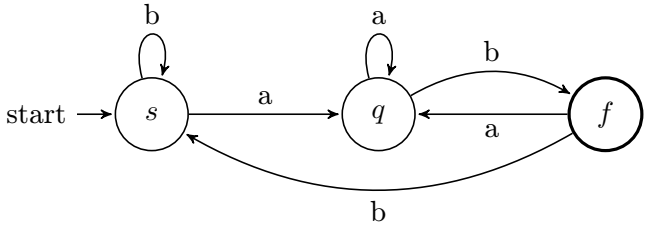
Definice 2.2.1. *Konečný automat* je pětice $A = (Q, \Sigma, R, s, F)$, kde Q je konečná *množina stavů* automatu, Σ je *abeceda vstupních symbolů*, R je konečná *množina pravidel* ve tvaru $qv \rightarrow q'$, kde $q, q' \in Q$ a $v \in \Sigma \cup \{\varepsilon\}$, $s \in Q$ je *počáteční stav* automatu a $F \in Q$ je *množina koncových stavů* automatu.

Automat začíná zpracování vstupního řetězce v počátečním stavu. Čte řetězec po znaku zleva doprava a postupně aplikuje pravidla podle aktuálního stavu a vstupního a aktuálně čteného symbolu. Kombinaci aktuálního stavu a dosud nepřečteného vstupu budeme nazývat *konfigurací* automatu. Aplikaci pravidla z R budeme nazývat *přechod* automatu a budeme značit $\vdash$. Pokud například automat přejde ze stavu q do stavu q' za současného přečtení symbolu a ze vstupu, zapíšeme přechod jako $qaw \vdash q'w$, $w \in \Sigma^*$ je zbývající nepřečtený vstup. Automat provedením přechodu změní svou konfiguraci. Z nové konfigurace může opět přejít do jiné. Více přechodů provedených za sebou nazýváme *posloupnost přechodů*. Pokud existuje posloupnost přechodů, pomocí které může automat přejít ze stavu q do stavu q', můžeme použít zkrácený zápis $q \vdash^* q'$ nebo $q \vdash^+ q'$, pokud je aplikován nenulový počet pravidel. Pokud automat po přečtení vstupního řetězce skončí v nějakém stavu $f \in F$, řetězec je automatem přijat. Jazyk L definovaný automatem $A = (Q, \Sigma, R, s, F)$ je pak

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

Každý jazyk, který je definován konečným automatem je regulární. Zároveň pro každý regulární jazyk existuje konečný automat, který je jeho modelem.

Příklad 1. Pro reprezentaci konečného automatu lze s výhodou využít grafické znázornění. Následující schéma znázorňuje automat přijímající všechny řetězce nad abecedou $\Sigma = \{a, b\}$, které končí podřetězcem ab



Obrázek 2.1: Ukázka grafické reprezentace konečného automatu

2.3 Zásobníkové automaty

Definice 2.3.1. *Zásobníkový automat* je sedmice $A = (Q, \Sigma, \Gamma, R, s, \$, F)$, kde Q je konečná *množina stavů* automatu, Σ je *abeceda vstupních symbolů*, Γ je *abeceda zásobníkových symbolů*, R je konečná *množina pravidel* ve tvaru $Wqv \rightarrow wq'$, kde $q, q' \in Q$, $v \in \Sigma \cup \{\varepsilon\}$, $W \in \Gamma \cup \{\varepsilon\}$ a $w \in \Gamma^*$, $s \in Q$ je *počáteční stav* automatu, \$ je *počáteční zásobníkový symbol*, kde $\$ \notin \Sigma \cup \Gamma$, a $F \subseteq Q$ je *množina koncových stavů* automatu.

Základem zásobníkového automatu je konečný automat, navíc je ale k dispozici zásobník, jako potenciálně nekonečná paměť pro ukládání symbolů z Γ. Automat opět začíná zpracování v počátečním stavu a postupně aplikuje přechodová pravidla společně se čtením vstupních symbolů a, pokud po přečtení celého vstupu skončí v nějakém stavu $f \in F$, je řetězec přijat. Úspěšnost zpracování lze také definovat jako přečtení celého vstupu a současné vyprázdnění zásobníku, na jehož vrcholu se nakonec nachází symbol \$. Obě podmínky je možné kombinovat. Rozdílem od konečného automatu je použití zásobníku. Do zásobníku je možné během přechodu vkládat symboly, odebírat je, a to v opačném pořadí, než byly vkládány, nebo vrchol zásobníku měnit. Mějme přechodové pravidlo $Wqv \to wq'$. Automat ve stavu q nejdříve při aplikování pravidla kontroluje nejen, zda je na vstupu symbol v, ale také, zda je na vrcholu zásobníku symbol W – ε ja na vrcholu vždy. Následně teprve může přejít do stavu q', přečíst symbol ze vstupu a W na zásobníku nahradit řetězcem w. I konfigurace zásobníkového automatu je odlišná od konečného. Konfigurace zásobníkového automatu je řetězec ve tvaru $\Gamma^*Q\Sigma^*$. Přechody zapisujeme obdobně zásobníkovému automatu ve tvaru $xWqvy \vdash xwq'y$. Za sebou následující přechody opět označujeme jako posloupnost a často zapisujeme zkráceně pomocí $\vdash^*$ nebo $\vdash^+$. Pokud budeme za přijetí řetězce považovat jeho přečtení skončené v koncovém stavu a s prázdným zásobníkem, automat $A = (Q, \Sigma, \Gamma, R, s, \$, F)$ bude definovat jazyk L

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

Všimněme si, že symbol \$ nelze ze zásobníku odstranit, protože nepatří do zásobníkové abecedy a proto se nemůže objevit v žádném pravidle. Každý jazyk, který je definován zásobníkovým automatem je bezkontextový. Zároveň pro každý bezkontextový jazyk existuje zásobníkový automat, který je jeho modelem.

2.4 Čisté zásobníkové automaty

Definice 2.4.1. *Čistý zásobníkový automat* je šestice $A = (Q, \Sigma, R, s, \$, F)$, kde Q je konečná *množina stavů* automatu, Σ je *abeceda vstupních symbolů*, R je konečná *množina pravidel* ve tvaru $Wqv \to wq'$, kde $q, q' \in Q$, $v \in \Sigma \cup \{\varepsilon\}$, $W \in \Sigma \cup \{\varepsilon\}$ a $w \in \Sigma^*$, $s \in Q$ je *počáteční stav* automatu, \$ je *počáteční zásobníkový symbol*, kde $\$ \notin \Sigma$, a $F \in Q$ je *množina koncových stavů* automatu.

Čisté zásobníkové automaty se od zásobníkových automatů příliš neliší, proto se podíváme blíže pouze na rozdíly mezi nimi. Veškeré rozdíly mezi nimi plynou ze skutečnosti, že čisté zásobníkové automaty nemají zvlášť definovanou zásobníkovou abecedu Γ, protože $\Gamma \subseteq \Sigma$. Na zásobníku se budou objevovat pouze symboly ze vstupní abecedy. V případě, že bude mít vstupní abeceda alespoň dva symboly, je možné pomocí nich zakódovat jakékoli množství symbolů (podrobný důkaz můžete nalézt například v [7]), proto omezení zásobníků na čisté ještě samo nesnižuje výpočetní sílu zásobníkových automatů. Uveďme si příklad čistého zásobníkového automatu.

Příklad 2. Mějme automat A

$$A = (Q, \Sigma, R, \$, s, F),$$
$$Q = \{s, f\},$$

$$
\begin{aligned}
\Sigma &= \{a, b\}, \\
R &= \{sa \ \ \rightarrow as, \\
&\quad\ \ asb \rightarrow f, \\
&\quad\ \ afb \rightarrow f \\
&\quad\ \}, \\
F &= \{f\}
\end{aligned}
$$

Definovaný automat je modelem pro jazyk $a^k b^k$, kde $k \in \mathbb{N}$. Ilustrujme si nyní výpočet, který automat provede, pokud bude mít na vstupu řetězec *aaabbb*

1. $\$saaabbb \vdash \$asaabbb$
2. $\$asaabbb \vdash \$aasabbb$
3. $\$aasabbb \vdash \$aaasbbb$
4. $\$aaasbbb \vdash \$aafbb$
5. $\$aafbb \vdash \$afb$
6. $\$afb \vdash \$f$

Automat přečetl celý řetězec na vstupu, skončil v koncovém stavu a zároveň vyprázdnil zásobník. Řetězec je přijat.

Kapitola 3

Automaty s ohraničeným počtem čistých zásobníků a jejich vlastnosti

V sekci 2.4 jsme si již ukázali, jak fungují čisté zásobníkové automaty. Nyní přidáme některá omezení a budeme zkoumat vlastnosti čistých zásobníkových automatů s více než jedním zásobníkem. Přesněji, budeme se zabývat deterministickými čistými zásobníkovými automaty s omezeným počtem zásobníků. Otázkou nedeterminismu a jeho důsledků se zabývat nebudeme.

3.1 Definice uvažovaného zásobníkového automatu

Definice 3.1.1. Mějme $n \in \mathbb{N}$. *Čistým zásobníkovým automatem s n zásobníky* pak nazveme uspořádanou šestici

$$A = \big(Q, \Sigma, R, \$^n, s, F\big)$$

kde Q je konečná množina všech *stavů* automatu, Σ je *abeceda vstupních symbolů*, R je *relace přechodu* automatu, $\$^n$ je *počáteční konfigurace zásobníků automatu*, kdy každý z n zásobníků obsahuje pouze speciální symbol $\$ \notin \Sigma$ označující jeho dno, $s \in Q$ je *počáteční stav* automatu a $F \subseteq Q$ je množina *koncových stavů*.

Před bližším upřesněním relace R je třeba zavést tzv. konfiguraci automatu. Konfigurace automatu je řetězec ve tvaru

$$\big(\Sigma^*\{\$\}\big)^n Q \Sigma^*$$

Skládá se z obsahu všech zásobníků automatu, stavu automatu a slova na vstupu. Konfiguraci budeme proto zapisovat

$$v_n\$ \ldots \$v_2\$v_1\$qw$$

kde $v_i\$$ jsou obsahy jednotlivých zásobníků ode dna vpravo až po vrchol vlevo, q je stav automatu a w je slovo na vstupu čtené zleva.

Definice 3.1.2. Relace R se nazývá *přechodová funkce*. Definuje pravidla pro přechody mezi jednotlivými konfiguracemi automatu. Automat přečte vstupní symbol nebo ε, přejde

ze stavu q do stavu q', $q, q' \in Q$, které mohou a nemusí být shodné, a může přidávat či odebírat symboly na vrcholech zásobníků. Pro deterministické automaty, kterými se budeme zabývat, se jedná o zobrazení mezi dvěma konfiguracemi, které budeme značit $\rightarrow$. Pravidla budeme zapisovat ve tvaru

$$\langle\varphi\rangle qv \rightarrow q'$$

kde $q, q' \in Q$ jsou výchozí a cílový stav, $v \in \Sigma \cup \{\varepsilon\}$ je přečtený symbol a φ určuje zásobníkovou akci

$$\varphi \in \{\varepsilon\} \cup \big(\{1, 2, \ldots, n\} \times \{pop, push\} \times \Sigma\big)$$

kde n je počet zásobníků automatu. Pravidlo *pop* odebírá a *push* přidává na i-tý zásobník daný symbol ze Σ. Pokud $\varphi = \varepsilon$, nebude se zásobníky provedena žádná akce.

Pro přechody budeme používat symbol $\vdash$. Po sobě následující přechody budeme nazývat sekvencí přechodů. Přijímaný jazyk pak můžeme definovat jako sekvenci přechodů začínající v počátečním stavu automatu a končící v některém z koncových stavů automatu, během které je zároveň přečten celý vstupní řetězec a na konci které byly vyprázdněny všechny zásobníky. Jazyk L přijímaný automatem $A = \{Q, \Sigma, R, \$^n, s, F\}$ pak definujeme jako

$$L(M) = \big\{w \mid w \in \Sigma^*, \$^n sw \vdash^* \$^n f, f \in F\big\}$$

Hlavní důraz bude kladen na zkoumání minimálního počtu nutných zásobníků v závislosti na zpracovávaném jazyku. Nejdříve zavedeme některá omezení. Důležitým omezením bude zúžení zobrazení R tak, aby automat musel v každém kroku číst symbol ze vstupu. Zúžíme R jako zobrazení

$$\langle\varphi\rangle qv \rightarrow q'$$

kde definujeme v pouze jako $v \in \Sigma$. Ostatní zůstává zavedené shodně jako v Definici 3.1.2. Tím zamezíme, aby mohl automat se svými zásobníky pracovat nezávisle na čtení vstupu, vyloučíme možnost tzv. ε-přechodů. Na základě Definice 3.1.2 dostáváme následující tři typy možných přechodových pravidel automatu

1. $\langle\varepsilon\rangle qv \rightarrow q'$
2. $\langle i, push, v'\rangle qv \rightarrow q'$
3. $\langle i, pop, v'\rangle qv \rightarrow q'$

Platí, že $q, q' \in Q$ mohou a nemusí být shodné. To samé platí pro $v, v' \in \Sigma$. Automat může přečíst vstupní symbol bez ovlivnění obsahu zásobníků, může přečíst vstupní symbol a zároveň zapsat jeden symbol na vrchol některého ze zásobníků nebo může spolu s přečtením vstupního symbolu odstranit symbol z vrcholu některého ze zásobníků. Pravidla neumožňují, aby přečtení symbolu ovlivnilo více zásobníků.

Protože budeme zkoumat pouze střídání různě dlouhých sekvencí stejných symbolů, můžeme problém zúžit pouze na binární abecedu

$$\Sigma = \{a, b\}$$

Jednotlivé sekvence pak budou odlišeny tak, že pokud bude sekvence složena ze symbolů a, následující a předchozí budou složeny ze symbolů b. Obdobně pro sekvenci symbolů b. Vstupní slovo pak bude ve tvaru

$$(a^+b^+)^*a^*$$

Ve vstupu se budou objevovat různě dlouhé iterace symbolů a jádrem činnosti automatu bude počítání symbolů v jednotlivých sekvencích a porovnávání jejich délek. Protože ale délky nemusí být předem známé, slouží zásobníky jako potenciálně nekonečná paměť pro zapamatování informace o jedné sekvenci pro potřeby následného porovnání s informacemi o jiné následující. Obecně v takovém případě budeme hovořit o závislosti mezi těmito dvěma podřetězci řetězce. Závislosti mohou vyjadřovat různé vlastnosti souvisejících částí řetězce, zde se ale zúžíme pouze na závislosti délek jednotlivých sekvencí stejných symbolů v řetězci a to konkrétně pouze na shodu délek. Uveďme nyní příklad automatu podle aktuální definice.

Příklad 3. Buď $n = 2$. Pak automat A bude

$$\begin{aligned}
A &= \big(Q, \Sigma, R, \$\$, q_0, F\big),\\
Q &= \{q_0, q_1, q_2, q_3\},\\
\Sigma &= \{a, b\},\\
R &= \big\{\langle 1, push, a\rangle q_0 a \to q_0\\
&\qquad \langle 2, push, b\rangle q_0 b \to q_1\\
&\qquad \langle 2, push, b\rangle q_1 b \to q_1\\
&\qquad \langle 1, pop, a\rangle q_1 a \to q_2\\
&\qquad \langle 1, pop, a\rangle q_2 a \to q_2\\
&\qquad \langle 2, pop, b\rangle q_2 b \to q_3\\
&\qquad \langle 2, pop, b\rangle q_3 b \to q_3\\
&\quad \big\},\\
F &= \{q_3\}
\end{aligned}$$

Definovaný automat je modelem pro jazyk $a^k b^l a^k b^l$, kde $k, l \in \mathbb{N}$. Ilustrujme si nyní výpočet, který automat provede, pokud bude mít na vstupu řetězec *aaabbaaabb*

1. $\$\$q_0aaabbaaabb \vdash \$a\$q_0aabbaaabb$
2. $\$a\$q_0aabbaaabb \vdash \$aa\$q_0abbaaabb$
3. $\$aa\$q_0abbaaabb \vdash \$aaa\$q_0bbaaabb$
4. $\$aaa\$q_0bbaaabb \vdash b\$aaa\$q_1baaabb$
5. $b\$aaa\$q_1baaabb \vdash bb\$aaa\$q_1aaabb$
6. $bb\$aaa\$q_1aaabb \vdash bb\$aa\$q_2aabb$
7. $bb\$aa\$q_2aabb \vdash bb\$a\$q_2abb$
8. $bb\$a\$q_2abb \vdash bb\$\$q_2bb$
9. $bb\$\$q_2bb \vdash b\$\$q_3b$
10. $b\$\$q_3b \vdash \$\$q_3$

Automat přečetl celý řetězec na vstupu, skončil v koncovém stavu a zároveň vyprázdnil všechny zásobníky. Řetězec je přijat.

3.2 Zpracovávané jazyky a závislosti v nich

Dostatečným přijímacím modelem pro regulární jazyky je konečný automat a žádné zásobníky nejsou potřeba. Regulárními jazyky se zde proto nebudeme zabývat. Jazyky bezkontextové, které nejsou regulární, lze zpracovávat zásobníkovým automatem s jediným

zásobníkem. Jejich problematika se nás bude dotýkat, ale pouze okrajově. Ve středu pozornosti se budou nacházet především jazyky, které nejsou bezkontextové. Vstupní jazyk budeme zapisovat ve tvaru

$$a^{k_1}b^{k_2}a^{k_3}b^{k_4}a^{k_5}\dots$$

kde $k_i \geq 1$ a hodnota i značí pořadí dané sekvence v rámci řetězce. Tato forma zápisu vstupního jazyka bude pro další postup velmi důležitá. Z hlediska svých vlastností nás budou zajímat jazyky obsahující závislosti typu shoda indexů. Omezíme se na závislost dvojic indexů. Přesněji uvažujeme jazyky, kde platí

1. $\exists k_i \exists k_j : k_i = k_j \wedge i \neq j$
2. $\forall k_i \forall k_j \forall k_l : \big(k_i = k_j = k_l \wedge i \neq j \wedge i \neq l\big) \Rightarrow j = l$

Pro zpracování začínají být zajímavé až jazyky obsahující dvě a více takových závislostí. Zaměříme se především na strukturu vycházející z výše zmíněné formy jejich popisu.

Definice 3.2.1. Každá závislost se skládá ze dvou oddělených iterací symbolů vstupní abecedy s indexy k_i a k_j, pro které platí

$$k_i = k_j \wedge i < j$$

Sekvenci s indexem k_i budeme nazývat *výpočetní fáze* a sekvenci s indexem k_j *kontrolní fáze* závislosti. Závislost ve zpracovávaném jazyku může vypadat například následovně

Obrázek 3.1: Ukázka závislosti

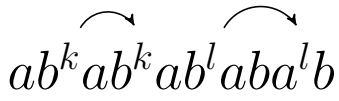
Z definice zároveň vyplývá, že se obě fáze závislosti nepřekrývají a navíc výpočetní fáze vždy předchází fázi kontrolní. Důvodem označení obou fází je jejich význam při zpracování automatem. Aby mohl kontrolovat shodu délek dvou oddělených sekvencí symbolů v průběhu jednosměrného průchodu, musí nejdříve vypočítat délku první z nich, zapamatovat si ji a následně zkontrolovat, zda se shoduje s délkou druhé z nich. Postup bude přesněji rozebrán později.

Definice 3.2.2. Mějme řetězec w obsahující m vzájemně nezávislých závislostí, které nazveme $z_1 \dots z_m$ po řadě podle pozice výskytu jejich výpočetní fáze v řetězci. Pro každou závislost z_x lze nalézt podřetězec představující část výpočetní fáze a podřetězec představující část kontrolní fáze v rámci jejího zpracování. Ve výše zmíněném popisu jazyka těmto dvěma fázím náleží dva indexy, které označíme k_i a k_j. Každá dvojice takových indexů náleží právě jedné závislosti v řetězcích zpracovávaného jazyka. Indexy k_i a k_j náležící závislosti z_x budeme označovat $k_{i_{z_x}}$ a $k_{j_{z_x}}$. Pokud nyní vezmeme všechny dvojice indexů náležící postupně závislostem $z_1 \dots z_m$, výsledkem bude množina

$$\mathcal{Z} = \left\{ \left(k_{i_{z_1}}, k_{j_{z_1}}\right), \left(k_{i_{z_2}}, k_{j_{z_2}}\right), \left(k_{i_{z_3}}, k_{j_{z_3}}\right), \ldots, \left(k_{i_{z_m}}, k_{j_{z_m}}\right) \right\}$$

Nyní můžeme identifikovat několik jevů, které budeme v souvislosti s výskytem závislostí v řetězci zkoumat. Mějme dvě závislosti z_x a z_y a k nim příslušné sekvence symbolů náležící jejich výpočetním i kontrolním fázím po řadě s dvojicemi indexů $(k_{i_{z_x}}, k_{j_{z_x}}), (k_{i_{z_y}}, k_{j_{z_y}}) \in \mathcal{Z}$. Že se závislosti *nepřekrývají*, budeme tvrdit o situaci, kdy platí

$$j_{z_x} < i_{z_y}$$



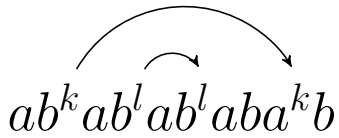
Obrázek 3.2: Závislosti, které se nepřekrývají

Překrýváním závislostí pak budeme označovat situaci, kdy platí

$$i_{z_x} < i_{z_y} \wedge j_{z_x} > i_{z_y}$$

Následně budeme rozlišovat dva typy překrývání. *Zanořením* závislostí budeme označovat případ, kdy navíc platí

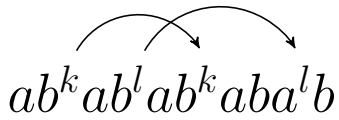
$$j_{z_x} > j_{z_y}$$



Obrázek 3.3: Zanoření závislostí

Naopak za *křížení* závislostí budeme považovat překrývání, při kterém platí

$$j_{z_x} < j_{z_y}$$



Obrázek 3.4: Křížení závislostí

Je zřejmé, že jiný typ překrývání závislostí neexistuje. Jazyky bez překrývání závislostí, nebo obsahující pouze překrývání typu zanoření je možné zpracovávat jediným zásobníkem, podobně jako obvyklý závorkový jazyk. Zajímavější proto pro nás budou jazyky obsahující překrývání typu křížení závislostí.

3.3 Postup zpracování vstupního řetězce automatem

Definice funkce automatu musí respektovat požadavky dané zpracovávaným jazykem za předpokladu dodržení definovaných omezení automatu. Automat musí být schopen zaznamenat délku libovolně dlouhé sekvence symbolů. Zároveň musí být schopen zaznamenanou délku sekvence porovnat s délkou další následující sekvence symbolů. Automat zároveň při výběru přechodového pravidla vychází pouze z aktuálního stavu, prvního symbolu na vstupu a symbolů na vrcholech jednotlivých zásobníků.

Definice 3.3.1. Buď S množinou všech přípustných délek sekvence symbolů v rámci závislosti – zde $S = \mathbb{N}$, obecně může ale být jazyk definován s určitým omezením délky. Buď

$$\mathcal{K} = \left\{ k \mid k \in \left(\Sigma^*\{\$\}\right)^n Q \right\}$$

Pro zpracování výpočetní fáze každé závislosti musí automat definovat relaci $\longmapsto$, kterou budeme nazývat *relací délky* a pro kterou platí

1. $\forall s \in S : \exists k \in \mathcal{K} : k \longmapsto s$
2. $\forall s_1 \forall s_2 \in S : \forall k \in \mathcal{K} : k \longmapsto s_1 \wedge k \longmapsto s_2 \Rightarrow s_1 = s_2$

Definice vyplývá z potřeby zaznamenat libovolnou délku sekvence a z následné potřeby ze zaznamenané informace délku zpětně jednoznačně určit. Jediná činnost automatu spočívá v přechodech mezi jeho konfiguracemi, je proto jediným možným prostředkem automatu, jak informaci o délce sekvence zaznamenat. V rámci přechodu může automat měnit svůj stav a vkládat nebo odebírat symboly ze zásobníků.

Věta 1. *Použití stavů automatu k zaznamenání délky sekvence nezvýší možnosti záznamu.*

Důkaz. Množina S není konečná. Protože definovaná relace délky každému jejímu prvku, jakožto obrazu, přiřazuje alespoň jeden vzor z množiny $\mathcal{K}$, ani ta nemůže být konečná. Tyto vzory vznikají kombinací stavu a obsahů všech zásobníků. Jejich počet je roven součinu $card(Q) \cdot card((\Sigma^*\{\$\})^n)$. Množina stavů je konečná, zatímco množina kombinací možných obsahů všech zásobníků není, $card((\Sigma^*\{\$\})^n) = \infty$. Víme že, pokud $card(Q)$ je přirozené číslo, $card(Q) \cdot \infty = \infty$. z toho vyplývá, že použití stavů automatu k zaznamenávání délky sekvence nezvýší počet možností záznamu, důkaz Věty 1 je hotov. □

Z Věty 1 vyplývá, že dále nemusíme změnu stavu automatu v rámci počítání délky sekvence symbolů uvažovat. Výpočet délky sekvence symbolů bude probíhat pouze za použití zásobníků automatu. Definujme si proto zásobníkovou konfiguraci jako aktuální obsah všech zásobníků automatu. Pro potřeby záznamu nás nebude zajímat kompletní konfigurace automatu, ale pouze jeho zásobníková konfigurace jako prvek množiny

$$\left(\Sigma^*\{\$\}\right)^n$$

Přechodem mezi zásobníkovými konfiguracemi pak budeme mít na mysli přechod mezi konfiguracemi automatu, v rámci jehož nás bude zajímat pouze změna zásobníkové konfigurace. Zároveň můžeme bez újmy na obecnosti zúžit množinu vzorů relace délky. Buď

$$\mathcal{K} = \{k \mid k \in (\Sigma^*\{\$\})^n\}$$

Načítání sekvence probíhá po jednotlivých znacích a v jeho průběhu automat nemůže předem odhadnout její délku. s každým načteným symbolem musí proto automat přejít do zásobníkové konfigurace, která je v relaci délky pouze s právě přečtenou délkou sekvence. Zároveň definice automatu nabízí tři typy přechodů, které automat může provést.

Možnost použití prvního typu přechodů definice relace délky snadno vyloučí. Při takovém přechodu se nemění zásobníková konfigurace automatu, čímž se stejná zásobníková konfigurace dostává do relace délky se dvěma různými délkami sekvence symbolů v rámci aktuálně zpracovávané závislosti. To Definice 3.3.1 vylučuje.

Samostatné použití přechodů třetího typu pro záznam délky sekvence můžeme také snadno zavrhnout. Počet různých délek, kterých může zpracovávaná sekvence nabývat, není konečný počet, celkový počet všech symbolů uložených na zásobníky automatu je ale vždy konečný, proto tento postup opět neumožňuje definovat relaci délky.

Vyloučení kombinace přechodů druhého a třetího typu je třeba rozdělit na dvě části. Pokud bychom při provádění přechodu třetího typu v rámci zpracování výpočetní fáze závislosti odebírali symboly, které byly uloženy ještě před zpracováním této závislosti, narážíme na jejich konečný počet, zatímco relace délky vyžaduje nekonečný prostředek záznamu. Pokud bychom naopak odebírali symboly přidané při zpracovávání aktuální závislosti a zároveň zachovávali podmínku, že musí být každá zásobníková konfigurace v rámci zpracování sekvence jedinečná, můžeme odebíráním prokládat vkládání, které musí převažovat, abychom se odebráním symbolu nevrátili do dřívější zásobníkové konfigurace. Tento postup může být v souladu s předchozími podmínkami. Na rozdíl od čistého vkládání symbolů je možné dosáhnout například úspornějšího záznamu v závislosti na definovaném prokládání, ale nemůžeme zvýšit počet možností záznamu. Důvod je jednoduchý, opět platí $k \cdot \infty = \infty$, pokud k je kladná konstanta, zde prokládání. Jako prostředek záznamu budeme proto uvažovat pouze přechodová pravidla druhého typu, vkládání symbolů.

Zaměříme se na druhou část zpracování závislosti, kontrolu shody délek zaznamenané sekvence a k ní příslušné následující. Předpokládejme, že má automat na vstupu slovo, které vyhovuje definici jazyka, který přijímá. Pak bude přečtení obou částí závislosti rozloženo na shodné množství přechodů automatu. Automat si délku první sekvence symbolů zaznamenal uložením stejného počtu symbolů. Informace v této formě musí následně sloužit při porovnávání. Automat v rámci čtení sekvence kontrolní fáze závislosti potřebuje uložené symboly přečíst, čímž zkontroluje shodu délek obou sekvencí. Čtení proběhne pomocí odstraňování symbolů ze zásobníku. Přechody vkládající symbol na zásobník, nebo nemanipulující se zásobníkem z této fáze musíme vyloučit. Důvodem je, že na přečtení uložených symbolů z výpočetní fáze je třeba využít celou kontrolní fázi. Při čtení každého symbolu ze vstupu bude odstraněn jeden ze zásobníku, kterých je stejný počet.

Ze zásobníkového přístupu a postupů záznamu a kontroly délky závislosti pak vyplývá následující tvrzení. Zavádí dva nutné předpoklady zpracování závislosti.

Věta 2. *Automat musí být definován tak, aby vždy zajistil, že při čtení prvního symbolu sekvence náležícího kontrolní fázi každé závislosti bude na vrcholu některého ze zásobníků poslední symbol zapsaný v rámci související výpočetní fáze. Při čtení každého následujícího symbolu této sekvence, mimo posledního, pak musí být na vrcholu některého ze zásobníků symbol předcházející poslednímu symbolu přečtenému ze zásobníku.* □

Nyní je třeba určit, jak toho automat docílí. Nejdříve se zaměříme na druhou část předpokladu. Aby byl po odstranění symbolu ze zásobníku další dříve uložený symbol na vrcholu některého ze zásobníků, může být buď na vrcholu jiného zásobníku ještě před odstraněním symbolu, nebo musí být uložen na stejném zásobníku hned před odstraněným symbolem. Jiná možnost nepřipadá v úvahu, protože v rámci tohoto přechodu už není možné obsahy zásobníků nijak jinak ovlivnit. Možnost, že by se následující symbol nacházel na vrcholu jiného zásobníku můžeme vyloučit, protože zásobníků má automat konstantní počet, symbolů v sekvenci může ale být potenciálně více. Proto zbývá jako vhodné a zjevně jednodušší řešení symboly v rámci výpočetní fáze ukládat na stejném zásobníku jeden po druhém v sekvenci. Pak je možné je stejně snadno, pouze v opačném pořadí, ze zásobníku číst a bude vždy zajištěna platnost druhé části zavedeného předpokladu.

První předpoklad už se těsně dotýká klíčového problému celé této práce. Budeme se jím zabývat v následující sekci.

3.4 Zpracování jazyků obsahujících více závislostí

V předchozích sekcích jsme již rozebrali, jak zpracovávat definovaným automatem závislost. Nyní se zaměříme na zpracování jazyka obsahujícího více závislostí. Nejdříve nás bude zajímat současné zpracování dvou závislostí. Dvojice závislostí jsme rozdělili podle morfologie jejich vzájemného rozložení do tří tříd. Hlavním předmětem našeho zkoumání bude vztah zpracovávaného jazyka z pohledu rozložení jeho závislostí a nutného minimálního počtu zásobníků, které automat musí mít k dispozici, aby byl zpracování schopen. Proto dříve než rozebereme postupy automatu při zpracování více závislostí, je nutné zavést pro nás klíčový předpoklad. v tomto textu nás budou dále zajímat automaty, které si definujeme jako automaty s minimálním nutným počtem zásobníků.

Definice 3.4.1. Mějme $n \in \mathbb{N}$ a automat A s n zásobníky definující jazyk J. A je *automatem s minimálním nutným počtem zásobníků*, pokud neexistuje žádný automat A' s $n-1$ zásobníky, který by jazyk J definoval.

Jinak řečeno, budeme hledat hranice, kdy je ještě daný jazyk možné automatem zpracovávat a kdy už ne, vše v souladu se zavedenými omezeními a předpoklady. Základním prvkem úsporného zpracování je maximální využití potenciálu každého ze zásobníků. Pro zpracování každé závislosti je třeba uložit informaci, proto je vždy třeba využít alespoň jeden zásobník. Nejdříve identifikujeme případy, kdy je jeden zásobník zároveň dostatečným.

Mějme jazyk obsahující n závislostí, které se nepřekrývají. Podle postupu definovaného v předchozí sekci bude automat u každé z nich postupovat následovně. Nejprve po jednom symbolu načte na zásobník sekvenci v délce celé výpočetní fáze závislosti. Následně po jednom symbolu celou sekvenci ze zásobníku odstraní v rámci zpracování kontrolní fáze závislosti. Výsledkem je, že zásobník zůstává na konci prázdný. Předpokládáme ovšem, že na začátku kontrolní fáze zpracování závislosti bude na vrcholu zásobníku poslední symbol uložený v předchozí související výpočetní fázi. Protože se závislosti nepřekrývají, se zásobníkem se mezi oběma fázemi každé z nich nijak nepracuje a předpoklad proto nutně zůstává splněn. Každá závislost proto po svém dokončení uvede zásobník do původní zásobníkové konfigurace – zde vyprázdní – a následující může proběhnout opět stejným způsobem. v tomto případě automatu stačí jediný zásobník pro zpracování všech n závislostí i za předpokladu, že n je libovolně velké.

Mějme jazyk obsahující n závislostí, které se překrývají, ale nekříží – jsou zanořené. Výpočetní a kontrolní fáze každé závislosti mohou být z hlediska práce se zásobníkem oddělené. Mezi fázemi s ním může být manipulováno. Automat ale musí zajistit, aby při započetí kontrolní fáze byl na vrcholu zásobníku poslední symbol uložený v rámci související výpočetní fáze. Protože se ale závislosti nekříží, pokud po výpočetní fázi některé z nich proběhne výpočetní fáze další, bude výskyt jejich kontrolních fází v opačném pořadí. Proto symboly později uložené budou ze zásobníku dříve odstraněny a každá kontrolní fáze bude pracovat se souvisejícími daty. Takto může automat opět zpracovat všech n závislostí použitím jediného zásobníku pro libovolně velké n.

Bylo by ještě vhodné zmínit, jak automat na jediném zásobníku rozpoznává hranice jednotlivých sekvencí. Protože pracujeme s binární abecedou, nejjednodušším postupem je zapisovat každou sekvenci celou pomocí stejných symbolů a za sebou uložené sekvence odlišit použitím různých symbolů – pokud je sekvence složena ze symbolů a, následující bude ze symbolů b.

Pokud jazyk obsahuje jak sekvence, které se nepřekrývají, tak sekvence zanořené, automat kombinuje oba výše zmíněné postupy. Stále je jediný zásobník dostatečným nezávisle na počtu závislostí. Nedostatečným se stává až v případě křížení závislostí. Zásobníkový přístup je typu LIFO (z anglického Last In First Out), což je omezení, se kterým se zpracování musí vypořádat. Definujme si termíny doba výskytu a prodleva závislosti.

Definice 3.4.2. Mějme řetězec w obsahující závislost z, jako rovnost délek sekvencí symbolů $z_1, z_2 \in \Sigma^*$

$$w = vz_1^k x z_2^k y$$

kde $v, x, y \in \Sigma^*$. Pak *doba výskytu* závislosti z bude $|v|$ a *prodleva* závislosti z bude $|x|$.

Věta 3. *Pokud se dvě závislosti kříží, nemohou ke svému zpracování použít stejný zásobník.*

Přesněji, mějme závislost z s indexem k, dobou výskytu d a prodlevou p. Mějme závislost z' s indexem k', dobou výskytu d' a prodlevou p'. Buď $d < d' < d + k + p$. Pokud $d + k + p < d' + 2 \cdot k' + p'$, závislost z' nesmí použít v rámci své výpočetní fáze stejný zásobník.

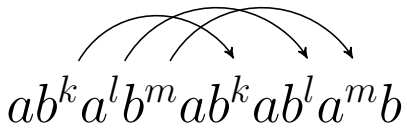
Důkaz. Mějme dvě křížící se závislosti. Předpokládejme, že Věta 3 neplatí, tedy že závislosti mohou pro své zpracování použít stejný zásobník. Nejdříve proběhne výpočetní fáze první a na zásobník je uložena příslušná sekvence symbolů. Ta musí být opět na vrcholu zásobníku, až se na vstup dostane související sekvence příslušná kontrolní fázi v rámci stejné závislosti. Před touto kontrolní fází ale přichází na vstup ještě výpočetní fáze další závislosti. Opět je třeba na zásobník uložit sekvenci symbolů. Ta bude ale odstraněna v rámci kontrolní fáze, která nastane až po kontrolní fázi první závislosti. Nastává spor s principem zásobníkového přístupu. Předpoklad neplatí, musí proto platit Věta 3. □

Jednodušeji, zásobník, který už obsahuje nějaké uložené informace, může být pro uložení dalších využit pouze v případě, že později uložené informace budou vyžadovány dříve než dříve uložené – princip LIFO. Problém dvou křížících se závislostí má za podmínky dodržení tohoto předpokladu jednoznačné vyústění. Druhá závislost již nemůže využívat stejný zásobník jako první. Proto bude pro zpracování nutné použít minimálně dva zásobníky. Zaveďme pojem n-násobného křížení.

Definice 3.4.3. Mějme $n-1$ závislostí z_i s indexy k_i, dobami výskytu d_i a prodlevami p_i, $1 \leq i \leq n-1$, které se vzájemně kříží. Mějme závislost z_n s indexem k_n, dobou výskytu d_n a prodlevou p_n. Pokud současně platí

1. $\forall d_i, 1 \leq i \leq n-1 : d_i < d_n$
2. $\forall d_i, k_i, p_i, 1 \leq i \leq n-1 : d_i + 2 \cdot k_i + p_i > d_n$
3. $\forall d_i, k_i, p_i, 1 \leq i \leq n-1 : d_i + 2 \cdot k_i + p_i < d_n + k_n + p_n$

pak závislost z_n kříží všechny závislosti $z_1 \ldots z_{n-1}$. Jedná se o *n-násobné křížení.*



Obrázek 3.5: Příklad trojnásobného křížení

Věta 4. *Mějme jazyk obsahující alespoň jedno n-násobné křížení, kde $n \geq 2$. Čistý zásobníkový automat ke zpracování jazyka potřebuje tolik zásobníků, kolik je maximální hodnota křížení n ze všech n-násobných křížení.*

Důkaz. Důkaz provedeme indukcí pro $n \geq 2$, kde n je počet maximálně se současně křížících závislostí.

Indukční základ. Buď $n = 2$. Pak platí Věta 3. Aby automat mohl definovat takový jazyk, potřebuje alespoň dva zásobníky.

Indukční hypotéza. Předpokládejme, že Věta 4 platí pro nějaké $n \geq 2$.

Indukční krok. Mějme jazyk, který obsahuje n současně se křížících závislostí, pro které platí indukční hypotéza. Automat potřebuje alespoň n zásobníků, aby byl schopný takový jazyk přijímat. Změníme definici jazyka tak, aby obsahoval navíc jednu závislost, která bude křížit všech n současných. Jazyk bude obsahovat $n+1$ současně se křížících závislostí. Pokud nyní po řadě aplikujeme Větu 3 na všech n dvojic, které původní závislosti s nově přidanou vytvoří, zjistíme, že žádné dvě nemohou použít pro zpracování stejný zásobník. Automat pro zpracování $n+1$ současně se křížících závislostí potřebuje jeden zásobník navíc, než pro zpracování n současně se křížících závislostí. Tím je důkaz hotov. □

Nyní se nám podařilo na základě morfologie rozložení závislostí v definici jazyka nalézt nekonečnou hierarchii jazyků, které korespondují k automatům s omezeným počtem čistých zásobníků. Každá vyšší třída jazyků v této hierarchii vyžaduje automat s vyšším počtem zásobníků, než vyžadují třídy nižší.

3.5 Korespondence k Chomského hierarchii

Nabízí se zkoumat, zda jazyky, které automaty z 3.1 definují, korespondují nějakým způsobem k Chomského hierarchii jazyků. Jak jsme předvedli v příkladu 3, automaty s ohraničeným počtem čistých zásobníků jsou schopné definovat jazyky, které nejsou bezkontextové. Ukážeme si ale, že zároveň nedokáží zpracovávat některé bezkontextové jazyky.

Příklad 4. Mějme jazyk $J = a^m b^n$, $m > n$. Pokud je jazyk J bezkontextový, musí existovat zásobníkový automat, který jej definuje. Mějme zásobníkový automat A

$$
\begin{aligned}
A &= \big(Q, \Sigma, \Gamma, R, \$, s, F\big),\\
Q &= \{s, q, f\},\\
\Sigma &= \{a, b\},\\
\Gamma &= \{a\},\\
R &= \{sa \;\; \to as,\\
&\quad\; asb \to q,\\
&\quad\; aqb \to q\\
&\quad\; aq \;\; \to f\\
&\quad\; af \;\; \to f\\
&\quad\},\\
F &= \{f\}
\end{aligned}
$$

Automat A jazyk J definuje. Nejdříve čte symboly a ze vstupu a ukládá je na zásobník. Následně čte symboly b a odebírá ze zásobníku a. Pokud je sekvence symbolů b kratší, než sekvence symbolů a, zůstane na zásobníku po přečtení řetězce ještě alespoň jeden symbol a a automat pomocí ε-přechodu přejde do koncového stavu f. V tomto stavu případně ještě vyprázdní zásobník a tím je řetězec přijat. Pokud by ale nebyla splněna podmínka jazyka J a sekvence symbolů b by nebyla kratší, po vyprázdnění zásobníku by nebylo možné použít žádné další pravidlo a přejít do koncového stavu. Řetězec by přijat nebyl. Automat proto přijímá pouze sekvence symbolů a následované kratší sekvencí symbolů b, což je přesně definice jazyka J.

Příklad 4 ukazuje, že jazyk $a^m b^n$, $m > n$, je bezkontextový. Pokud bychom se ho ale snažili přijímat automatem, který jsme definovali v 3.1, neuspěli bychom. Důvodem je, že jsme vyloučili použití ε-přechodů. Proto by nebylo možné na konci po porovnání délek sekvencí vyprázdnit zásobník od zbylých symbolů a řetězec by proto nemohl být přijat.

Třída jazyků definovaná zkoumanými automaty nekoresponduje s žádnou třídou jazyků Chomského hierarchie. Vlastní nadtřídou je pouze pro jazyky regulární. Bylo by proto vhodné dále zkoumat, jak by se změnila třída definovaných jazyků, kdybychom některá zavedená omezení zrušili. Základním omezením automatů jsou jejich čisté zásobníky, jak bylo ale zmíněno v 2.4, samo omezení nemá vliv na výpočetní sílu automatů. Navíc, pokud bychom omezení zrušili, zkoumali bychom již výrazně odlišné formální modely. Hlavním omezením zůstává proto podoba definice přechodové funkce R, konkrétně absence ε-přechodů. V příkladu 4 jsme narazili na bezkontextový jazyk, který ale zásobníkový automat bez ε-přechodů nedokáže přijímat. Podíváme se, jak by se změnila výpočetní síla automatu, kdybychom předefinovali přechodovou funkci. Buď přechodová funkce R obsahující pravidla tvaru

$$\langle\varphi\rangle qv \to q'$$

kde φ ponecháme definované shodně jako v 3.1, $q, q' \in Q$ a $v \in \Sigma \cup \{\varepsilon\}$. Nyní již automat může provést přechod, aniž by musel přečíst symbol ze vstupu. Může manipulovat se zásobníkem nezávisle na čtení vstupu. Výpočetní sílu výsledné třídy modelů budeme nyní zkoumat v závislosti na počtu zásobníků, kterými automat disponuje. Když víme, že čisté zásobníky nesnižují výpočetní sílu automatů, třída jazyků definovaná automaty s jediným čistým zásobníkem bude korespondovat k třídě jazyků definovaných zásobníkovými automaty, třídě bezkontextových jazyků. Zásobníkové automaty s alespoň dvěma zásobníky jsou dostatečně silné, aby definovali celou třídu rekurzivně vyčíslitelných jazyků. Pohyb čtecí hlavy po vstupní pásce mohou simulovat přesunem symbolů z jednoho zásobníku na druhý a rozšiřitelnost pásky zajistí potenciálně nekonečná kapacita obou zásobníků (viz [6]). Předefinované zásobníkové automaty s alespoň dvěma čistými zásobníky budou nutně stejně silné. Dostáváme třídu automatů se silou Turingova stroje.

Kapitola 4

Další vývoj

Během našeho zkoumaní jsme u definovaných modelů objevili omezení nebo nedostatky. Problematické je především obtížné stanovení síly modelů vzhledem k již známým modelům. Proto by bylo vhodné zaměřit další zkoumání na jejich možné úpravy. Zároveň jsme jejich vlastnosti zkoumali na omezené třídě jazyků, bylo by vhodné zkoumání rozšířit.

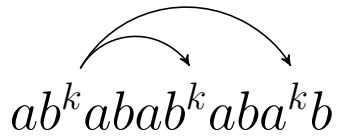
4.1 Možné úpravy modelů

Automaty definované v sekci 3.1 se ukázali jako problematické při pokusu o zasazení do Chomského hierarchie jazyků. Řešení problému pravděpodobně spočívá v předefinování přechodové funkce. Zde se ovšem nabízí mnoho možností a bylo by následně vhodné se jimi zabývat. Zásadní se ukázala absence ε-přechodů. Pouhé rozšíření definice přechodových pravidel o možnost ε-přechodů ale postaví automaty s čistými zásobníky do roviny se zásobníkovými automaty. Přechodová funkce by mohla být definována s omezeními či rozšířeními, které zatím uvažovány nebyly. Můžeme zavést například pojem *kompletního vyprázdnění* podobně jako v [4], případně zavést některá dodatečná omezení automatů například na počty přístupů do zásobníků nebo pořadí přístupu a podobně. Omezeními bychom mohli získat modely, které výpočetní silou korespondují k třídě kontextových jazyků. Vhodným rozšířením stávajících modelů by mohla být například současná práce s více zásobníky v rámci jediného přechodu. Také jsme se nezabývali případným vlivem nedeterminismu na sílu automatů. Množství variant, které by se daly uvažovat, by jistě bylo obrovské. Jejich vliv může být předmětem dalšího zkoumání.

4.2 Rozšíření zkoumaných závislostí

Po většinu této práce jsme se na vstupní jazyk dívali pohledem kombinatoriky na slovech. Pokud bychom chtěli dále rozšířit takto zkoumanou problematiku, určitě by bylo vhodné se zaměřit na vstupní jazyk. Definovali jsme závislost a omezili ji pouze na shodu dvou indexů. Závislostí ale můžeme rozumět nejen shodu, může jí být nerovnost, násobek, mocnina a další. Obecně můžeme hovořit o matematickém vztahu popsaném rovnicí či nerovnicí. Zatím jsme ale nestanovili, závislosti jakého typu ještě automaty s omezeným počtem čistých zásobníků jsou schopné zpracovávat a na co už jejich výpočetní síla nestačí. Závislost jsme definovali pouze mezi dvěma sekvencemi symbolů v řetězci, ale jistě můžeme uvažovat

i vícenásobné závislosti, například shodu délek tří sekvencí symbolů. Opět nevíme, zda v takových případech vystačíme s definovanými modely, nebo zda je bude nutné upravit či například přidat zásobníky.



Obrázek 4.1: Příklad trojnásobné závislosti

Nakonec také můžeme rozšířit pojem závislosti ze vztahu dvou sekvencí znaků na obecnější vztah dvou řetězců. Můžeme ve vstupním jazyku například uvažovat iteraci řetězce symbolů. Ukazuje se, že dalšímu zkoumání tato oblast poskytuje ještě dostatečný prostor.

Kapitola 5

Závěr

V této práci byly definovány a zkoumány automaty s ohraničeným počtem čistých zásobníků. Dále byl zaveden termín závislosti. Zpracování vstupu automatem bylo studováno především s ohledem na závislosti vyskytující se ve vstupním řetězci. Definovali jsme možnosti vzájemného rozložení závislostí a zobecnili je na libovolný počet závislostí. Následně jsme zkoumali postup automatu při zpracování řetězce se závislostmi podle jejich vzájemného rozložení v definovaném jazyku. Podařilo se definovat nekonečnou hierarchii jazyků korespondujících se zavedenými automaty podle počtu jejich zásobníků. Ukázalo se ale, že třída jazyků definovaných zásbníkovými automaty s omezeným počtem čistých zásobníků nekoresponduje k žádné třídě jazyků Chomského hierarchie a vlastní nadtřídou je pouze pro regulární jazyky. V kapitole 4 jsme následně diskutovali další možný vývoj. Zaprvé by bylo třeba zkoumat možné úpravy automatů, aby lépe korespondovaly k Chomského hierarchii jazyků. Zadruhé by bylo možné zkoumání pojmu závislosti dále prohloubit. Nakonec byla vytvořena aplikace na které je možné demonstrovat činnost zkoumaných automatů. Popis jejího použití a ovládání lze nalézt v příloze B

Literatura

[1] C++ Reference. [online], [cit. 2011-05-07].
URL http://www.cplusplus.com/reference/

[2] Ito, M.: *Algebraic Theory of Automata and Languages*. World Scientific Publishing Co. Pte. Ltd., 2004, ISBN 981-02-4727-3.

[3] Leupold, P.; Meduna, A.: Finitely Expandable Deep PDAs. In *Automata, Formal Languages and Algebraic Systems - Proceedings of AFLAS 2008*, Kyoto, Japan, říjen 2008, str. 113.

[4] Masopust, T.; Meduna, A.: On Pure Multi-Pushdown Automata that Perform Complete Pushdown Pops. *Acta Cybernetica*, ročník 19, 2009: s. 537–552, ISSN 0324-721X.

[5] Prata, S.: *Mistrovství v C++*. Brno: Computer Press, a.s., třetí vydání, 2007, ISBN 978-80-251-1749-1.

[6] Rich, E.: *Automata, Computability and Complexity*. US: Prentice Hall, 2008, ISBN 0-13-228806-0.

[7] Rozenberg, G.; Salomaa, A.: *Handbook of Formal Languages*, kapitola Combinatorics of Words. New York: Springer, první vydání, 1997, ISBN 3-540-60420-0.

[8] Salomaa, A.: *Formal Languages*. New York: Academic Press, Inc., 1973, Library of Congress Catalog Card Number 72-88356.

Dodatek A

Obsah CD

1. **README**	Stručná nápověda k obsahu CD	
2. **thesis.pdf**	Text práce	
3. **interpret**	Zdrojová složka s demonstračním programem	(překlad příkazem `make`)
4. **latex**	Zdrojová složka s textovou částí práce	(překlad příkazem `make`)

Dodatek B

Interpret definované třídy automatů

Abychom mohli demonstrovat vlastnosti třídy automatů definované v kapitole 3, byl napsán interpret, který podle zadané definice vytvoří abstraktní model automatu a následně simuluje jeho činnost při zpracování vstupního řetězce. Interpret je schopný postihnout libovolný automat ze třídy definovaných automatů. Jedná se o program napsaný v jazyce *C++* (podrobnosti o jazyku naleznete například v [5]) psaný podle normy *ISO/IEC 14882* z roku 1998 s využitím knihovny *STL* (bližší popis knihovny můžete nalézt na [1]). K jeho přeložení slouží *Makefile* ve zdrojové složce, stačí v příkazové řádce zadat příkaz *make*. Výsledkem je spustitelný soubor s názvem *interpret*. Překlad byl prováděn na systémech

1. Linux 2.6.31-23-generic Ubuntu GNU/Linux	g++ (GCC) 4.4.1
2. Linux 2.6.32.32 GNU/Linux	g++ (GCC) 4.4.6
3. FreeBSD 8.2-STABLE	g++ (GCC) 4.2.1

Při spuštění program očekává zadání souboru s definicí automatu. Program vytvoří model automatu na základě zadané definice, čte vstupní slovo, implicitně ze *standardního vstupu* (*stdin*), a mění stav modelu v souladu s definovaným automatem. Program pracuje implicitně v tichém režimu, kdy na konci simulace pouze vypíše informaci o úspěšnosti zpracování vstupního řetězce automatem. Výpis provádí na *standardní výstup* (*stdout*), pokud se vyskytne chyba, program ji vypíše na *standardní chybový výstup* (*stderr*). Po vypnutí tichého režimu program s každým zpracovaným symbolem vypíše pravidlo, které bylo modelem automatu aplikováno. Na vstupu, případně ve vstupním souboru, program očekává vstupní řetězec, který má automat zpracovat. Vstupním řetězcem může být libovolná posloupnost znaků, které byly v definici automatu definovány jako vstupní symboly. Pokud program narazí na jiné znaky, skutečnost oznámí uživateli a skončí. Výjimku tvoří tzv. *bílé znaky*. Bílými jsou nazývány znaky, které jako parametr standardní funkce jazyka *C isspace* vrátí pozitivní pravdivostní hodnotu (mezera, tabulátor, konec řádky atd). Chování programu je dále upřesňováno parametry spuštění.

Automat zpracovává vstupní řetězec po jednotlivých znacích se současnou aplikací přechodových pravidel až do té doby, dokud buď celý řetězec nepřečte, nebo nemůže aplikovat žádné další pravidlo. Za úspěšný výpočet je považován takový, kdy je přečten celý vstupní

řetězec, vyprázdněny všechny zásobníky automatu a zároveň automat skončil v některém z koncových stavů. Pak je řetězec přijat. Jakýkoli jiný případ je považován za neúspěšný výpočet a řetězec není přijat.

Stejně jako v textu nebyla zkoumána problematika nedeterminismu, ani interpret nepočítá s nedeterministickými přechody. Pokud se model dostane do stavu, ze kterého může společně s přečtením vstupního symbolu provést více než jeden přechod, program oznámí skutečnost uživateli a zpracování je ukončeno.

B.1 Volby programu

Běh interpretu je možné přizpůsobit potřebám zadáním požadovaných parametrů příkazové řádky. Parametry lze zadávat v libovolném pořadí, je však třeba dbát na dodržení závislostí mezi nimi. Program podporuje tyto parametry

`-h`	Výpis nápovědy
`-v`	Vypnutí tichého režimu
`-s`	Režim krokování – vyžaduje zadané parametry `-v` a `-i`
`-d X`	Zadání souboru s definicí, kde `X` je soubor s definicí – povinný parametr
`-i X`	Zadání vstupního souboru, kde `X` je vstupní soubor

Program pracuje implicitně v tichém režimu, kdy vypisuje pouze informaci, zda byl vstupní řetězec automatem přijat, či nikoli, případně chybová hlášení. Po vypnutí tichého režimu automat vypisuje s každým přečteným symbolem navíc přechodové pravidlo, které bylo spolu s jeho přečtením aplikováno. Režim krokování zajistí, že program bude před aplikací každého následujícího pravidla čekat na interakci uživatele – stisk klávesy *enter*. Krokování simulace může sloužit pro snadnější studium běhu automatu. Zadání souboru s definicí automatu je povinným parametrem, protože definice automatu je základem pro vytvoření jeho modelu. Pokud je zadán vstupní soubor, vstupní řetězec není čten ze standardního vstupu, ale ze zadaného souboru. Režim krokování vyžaduje zadání vstupního souboru, protože standardní vstup je použit pro interakci s uživatelem. Zároveň režim krokování vyžaduje vypnutí tichého režimu, protože v něm by krokování nemělo smysl.

V případě nekorektního zadání parametrů nebo zadání neznámých či neúplných parametrů program uživatele upozorní chybovým hlášením, ze kterého je možné chybu zjistit. V případě nejistoty se vždy nabízí výpis nápovědy.

B.2 Formát definice automatu

Základem pro práci interpretu je definice, podle které může vytvořit model automatu. Program očekává zadání souboru s definicí pomocí parametru příkazové řádky. Soubor s definicí očekává v kódování ASCII. Definice musí přesně dodržovat zavedená syntaktická a sémantická pravidla, jinak není přijata. Tuto skutečnost program oznámí výpisem chybového hlášení. Uveďme si příklad definice, na kterém si popíšeme její formát a možnosti. Vstupní soubor s definicí může vypadat například takto

```
 1  Automat = ( Q, Sigma, R, 2, s, F )
 2  Q = { s, q, f, stav }
 3  Sigma = { a, b, c }
 4  R = { <1,+,a>s|a -> s,
 5        <2,+,b>s|b -> q,
 6        <2,+,b>q|b -> q,
 7        <1,-,a>q|a -> f,
 8        <1,-,a>f|a -> f,
 9        <2,-,b>f|b -> f,
10        <2,-,b>f|c -> stav,
11             <>stav|c -> stav
12      }
13  F = { stav }
```

Obrázek B.1: Ukázka souboru s definicí automatu

Rozeberme si nyní syntax a sémantiku definice. V definici můžeme rozeznat pět za sebou následujících struktur tvaru `X = Y`, kde `X` je název struktury a může jím být libovolný alfanumerický řetězec a `Y` je definice struktury. První struktura je uspořádanou šesticí, její definice je uzavřena v kulatých závorkách a pořadí prvků definice je pevně dané. Zbývající čtyři struktury jsou množinami, jejich definice jsou uzavřené ve složených závorkách. Pojem množina bude dále označovat právě tyto čtyři struktury. Pořadí prvků v definicích jednotlivých množin není pevně stanovené a jeho změna nemá na výsledný automat žádný dopad. Znaménka `=` musí být z obou stran oddělena nenulovým počtem bílých znaků. Všech pět struktur musí být od sebe oddělených nenulovým počtem bílých znaků a jejich pořadí je pevně dané.

První struktura, uspořádaná šestice, je popis automatu. Její definice obsahuje šest prvků navzájem oddělených znakem `,` a volitelně ohraničených libovolným počtem bílých znaků. Čtvrtý prvek je celé číslo a určuje počet zásobníků automatu. Ostatní prvky jsou libovolné alfanumerické řetězce. První prvek je názvem první množiny, množiny stavů automatu. Druhý prvek je názvem druhé množiny, abecedy vstupních symbolů. Třetí prvek je názvem třetí množiny, množiny přechodových pravidel. Pátý prvek je počátečním stavem automatu a musí být obsažen v definici množiny stavů automatu. Šestý prvek je názvem čtvrté množiny, množiny koncových stavů automatu. Názvy množin se musí shodovat s názvy, které popis automatu definuje.

Každá ze čtyř množin obsahuje libovolný počet položek oddělených znakem `,` a volitelně ohraničených libovolným počtem bílých znaků. Množina stavů automatu obsahuje libovolné navzájem odlišné alfanumerické řetězce, stavy automatu. Abeceda vstupních symbolů obsahuje pouze jednotlivé alfanumerické znaky, vstupní symboly, opět platí, že musí být navzájem odlišné. Množina koncových stavů automatu obsahuje obdobně jako množina stavů automatu libovolné alfanumerické řetězce, podmínkou ale je, že každý z nich je zároveň obsažen v množině stavů automatu.

Složitější syntax mají prvky množiny přechodových pravidel. Základní strukturu přechodových pravidel můžeme popsat jako `X -> Y`. Oddělovač `->` musí být ohraničen libovolným ale nenulovým počtem bílých znaků. Část `Y` je libovolný řetězec definovaný v množině stavů automatu, cílový stav přechodového pravidla. Část `X` je složena ještě ze tří dílčích složek ve tvaru `<U>V|W`. Složka `U` je zásobníková akce pravidla. Zásobníková akce může být buď prázdný řetězec, v takovém případě žádná akce provedena nebude, nebo obsahuje tři prvky oddělené znakem `,` bez dalších bílých znaků. Prvním prvkem zásobníkové akce je celé číslo, které udává, kolikátý zásobník bude použit. Číslo musí být v rozmezí 1 až počet zásobníků automatu. Druhým prvkem zásobníkové akce je buď znak `+`, pokud má být akcí vkládání symbolu na zásobník, nebo znak `-`, pokud má být symbol ze zásobníku odebírán. Třetím prvkem zásobníkové akce je znak dříve definovaný jako vstupní symbol v abecedě vstupních symbolů, který udává, jaký symbol má být přidán na zásobník či ze zásobníku odebrán. Složka `V` první části přechodového pravidla je výchozí stav. Jedná se o libovolný řetězec dříve definovaný v množině stavů automatu. Složka `W` je vstupní symbol, který má být s aplikací pravidla přečten. Opět se musí jednat o symbol dříve definovaný v abecedě vstupních symbolů.

Sémantika definice je pevná a neměnná, syntax je ale definována částečně volně. Je možné volit libovolné názvy množin, stavů a libovolné znaky jako symboly vstupní abecedy, stejně jako libovolně měnit počet stavů, přechodových pravidel atd. Díky možnosti vkládání bílých znaků lze definici přehledně formátovat dle vlastního uvážení.