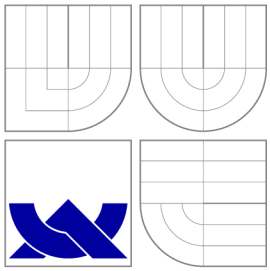
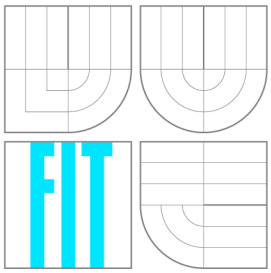


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYNTAKTICKÁ ANALÝZA ZALOŽENÁ NA GRAMATICKÝCH A AUTOMATOVÝCH SYSTÉMECH

PARSING BASED ON GRAMMAR AND AUTOMATA SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAKUB ŠOUSTAR
AUTHOR

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

BRNO 2015

Abstrakt

Tato práce se zabývá syntaktickou analýzou s využitím systémů paralelně komunikujících zásobníkových automatů. Zejména se zaměřuje na dopady nedeterminismu v jednotlivých komponentách na celý systém. Také je představen návrh algoritmu pro převod některých paralelně komunikujících gramatických systémů na systémy paralelně komunikujících zásobníkových automatů. Získané poznatky jsou použity při návrhu a implementaci metody syntaktické analýzy.

Abstract

This thesis is concerning with parsing using parallel communicating pushdown automata systems. Focusing especially on impacts of nondeterminism in individual components on the whole system. Also it introduces a proposal of algorithm for converting some parallel communicating grammar systems to parallel communicating pushdown automata systems. The gained knowledge is used to design and implement parsing method.

Klíčová slova

syntaktická analýza, paralelně komunikující, gramatické systémy, automatové systémy, nedeterminismus

Keywords

parsing, parallel communicating, grammar systems, automata systems, nondeterminism

Citace

Jakub Šoustar: Syntaktická analýza založená na
gramatických a automatových systémech, bakalářská práce, Brno, FIT VUT v Brně, 2015

Syntaktická analýza založená na gramatických a automatových systémech

Prohlášení

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

.....
Jakub Šoustar
13. května 2015

Poděkování

Chtěl bych poděkovat prof. RNDr. Alexandru Medunovi, CSc. za vedení a směřování této bakalářské práce, a zejména za jeho ochotu a cenné rady, které mi dal.

Obsah

Kapitola 1

Úvod

Lidé již od nepaměti používají pro dorozumívání se mezi sebou různých jazyků. Paleta těchto takzvaných přirozených jazyků je nesmírně pestrobarevná. S tím, jak se mění svět a lidé, se také mění jazyky, které používají. Významy slov se mění, vznikají různé nejednoznačnosti, dokonce i jazyky jako celky vznikají a zanikají.

S příchodem počítačů vyvstal problém, jak s nimi komunikovat. Přirozené jazyky, jakými se člověk dokáže bez větších problémů domluvit s jiným člověkem, jsou natolik komplikované, že pro potřeby jednoznačné komunikace s počítači jsou doposud nepoužitelné. Proto vznikly formální jazyky.

Formálním je takový jazyk, který jsme schopni přesně a jednoznačně matematicky popsat a definovat. Takové jazyky jsou vhodné pro komunikaci se stroji, neboť umožňují jednoduše určit přesný způsob, jak bude komunikace probíhat a jak bude vypadat. Můžeme jimi popsat programovací jazyky, datové formáty, přenosové protokoly a další.

Existuje také problém, jak formální jazyky popsat. Často totiž nechceme takový jazyk vyjádřit pouze jako konečnou množinu vět, které do něj spadají. Mnohem častější je požadavek, aby tyto jazyky byly nekonečné, to znamená, aby existoval neomezený počet vět, které do nich patří. Proto byly zavedeny gramatiky, které umožňují konečným množstvím pravidel přesně popsat, jak má vypadat nějaká věta, abychom o ní mohli říci, že patří do nějakého jazyka. Cenou za tuto přesnou definici je to, že jsou často schopny popsat pouze takové jazyky, jejichž vyjadřovací schopnost je v porovnání s jazyky přirozenými značně omezená. Proto již od doby jejich zavedení probíhá usilovný výzkum zaměřený na posílení jejich vyjadřovací síly.

Jedním z výsledků tohoto úsilí jsou gramatické systémy. Jejich základní myšlenkou je, že místo zavedení úplně nového modelu využívají již těch stávajících a spojují je do větších celků — systémů. Tyto systémy definují způsoby, jakými spolu jejich jednotlivé části spolupracují. Těmto částem říkáme komponenty systému. Příkladem spolupráce může být třeba výměna informací mezi dvěma a více komponentami. Obecně o takovýchto systémech můžeme říci, že jejich vyjadřovací schopnost přesahuje vyjadřovací schopnost jejich samotných komponent.

V současné době můžeme gramatické systémy rozdělit do dvou základních tříd lišících se způsoby, jak jejich komponenty mezi sebou spolupracují. Těmito dvěma základními druhy jsou systémy *sekvenční* a *paralelní*. Komponenty sekvenčních systémů spolupracují na tvorbě jednoho společného výsledku. V jednom okamžiku ovšem může takto pracovat pouze jedna komponenta, proto musí existovat centrální řídící prvek určující, kdy může která komponenta začít svoji práci. Druhým typem systémů jsou systémy paralelní. Zde již nepracují komponenty na jednom společném výsledku, ale každá pracuje samostatně na

svém vlastním. Komponenty mezi sebou mohou komunikovat a tak k výsledku své stávající práce přidat i výsledek ostatních komponent.

Syntaktickou analýzou je takový proces, kdy dochází k analýze nějaké věty s cílem zjistit, zda náleží příslušnému jazyku či ne. Obecným prostředkem pro realizaci této analýzy jsou takzvané automaty, které tvoří protějšek gramatik. Zatímco gramatiky jsou soubor pravidel říkajících, jak vytvořit nějakou větu jazyka, automaty jsou tvořeny souborem instrukcí, pomocí kterých je možné ověřit, zda nějaká věta patří do určitého jazyka. Říkáme také, že gramatiky jsou generativní model a automaty jsou akceptující model. Automaty i gramatiky spolu úzce souvisejí a tak se zpravidla nový objev v jedné oblasti projeví i v té druhé.

Nebylo tomu naopak ani v případě systémů spolupracujících automatů. Automatové systémy lze stejně jako ty gramatické rozdělit na dva základní druhy — *sekvenční* a *paralelní*. V sekvenčních systémech je vstup analyzován tak, že se postupně v práci střídají jednotlivé komponenty a každá zpracuje jeho určitou část. Program jejich střídání je opět daný řídící jednotkou. V paralelních systémech začnou všechny komponenty nezávisle na sobě zpracovávat kopie vstupu. Během této práce si mezi sebou mohou vyměňovat informace o dosavadním postupu jejich práce. Vstup potom systém přijme tehdy, přijmou-li jej všechny jeho části. Konkrétní druhy komunikace jsou u tohoto typu systémů závislé na použitých komponentách.

Gramatickým i automatovým sekvenčním systémům se říká *CD systémy (cooperating distributed)*. Pro paralelní systémy se používá označení *PC systémy (parallel communicating)*.

Cílem této práce je vytvoření takového syntaktického analyzátoru, který bude používat PC automatové systémy a který umožní přijmout i jazyky, které nejsou bezkontextové. Také představíme taková omezení pro PC gramatické systémy, která umožní jejich převod na PC automatové systémy akceptující stejný jazyk. Budeme uvažovat systémy jak deterministické tak i nedeterministické.

Cílem této práce je návrh a vytvoření takového syntaktického analyzátoru, který bude založen na PC automatových systémech a který umožní i analýzu takových jazyků, které nejsou bezkontextové. Postupně si podrobně představíme PC automatové systémy, uvedeme si některé obtíže, které je doprovázejí, představíme si možnosti, jak se s nimi vypořádat, včetně jedné konkrétní modifikace těchto systémů, a nakonec navrhneme metody syntaktické analýzy, která je na těchto systémech založená. Současně si také blíže představíme i PC gramatické systémy. Pro tyto systémy navrhneme taková omezení, která, jak věříme, umožní jejich převod na ekvivalentní PC automatový systém a tedy nám dovolí provést syntaktickou analýzu jazyků definovaných těmito systémy.

Kapitola 2

Základní definice a pojmy

V této úvodní kapitole si postupně shrneme některé důležité matematické pojmy, které budeme v této práci využívat. Následující definice i příklady jsme převzali z [7]. Pro podrobnější informace odkazujeme čtenáře na tuto knihu.

2.1 Množiny a n-tice

Množina M je soubor objektů, takzvaných prvků množiny, bez struktury, s vyjímkou členství. Pokud je prvek x členem M, značíme $x \in M$. Opačný výraz, tedy že x není členem M, zapisujeme jako $x \notin M$. Množinu, která nemá žádné členy, značíme $\emptyset$. Kardinalitou množiny M, psáno $card(M)$ nebo $|M|$, myslíme počet prvků M. Výrazem $|M|_E$ značíme počet výskytů prvku E v množině M. Množinu lze zapsat několika způsoby. Například množinu o třech prvcích lze zapsat jejich výčtem jako

$$M = \{a, b, c\}\,.$$

V případech, kdy je význam zřejmý, lze použít výpustek. Například $M = \{0, 1, \ldots, 9\}$ je množina čísel od nuly do devíti. V případě potřeby je možné vyjádřit množinu M charakteristickou vlastností π tak, že M obsahuje všechny prvky, které splňují vlastnost π. Pro tento způsob zápisu používáme tuto notaci

$$M = \{x \mid \pi(x)\}\,.$$

Základní operace, které s množinami provádíme, jsou *sjednocení* ($\cup$), *průnik* ($\cap$) a *rozdíl* ($-$). Mějme množiny M_1 a M_2. Potom

$$M_1 \cup M_2 = \{x \mid x \in M_1 \text{ nebo } x \in M_2\}$$
$$M_1 \cap M_2 = \{x \mid x \in M_1 \text{ a } x \in M_2\}$$
$$M_1 - M_2 = \{x \mid x \in M_1 \text{ a } x \notin M_2\}$$

Pokud je každý prvek množin N současně prvkem množiny M, N je *podmnožinou* M, značeno $N \subseteq M$. Zároveň M je *nadmnožinou* N. Pokud $N \subseteq M$ a současně $M - N \neq \emptyset$, N je *vlastní podmnožinou* M, psáno $N \subset M$. Také platí, že M je *vlastní nadmnožinou* N.

Potenční množina M, značená 2^M, je množina všech podmnožin M, tedy

$$2^M = \{U \mid U \subseteq M\}\,.$$

N-tice je seznam prvků, který, na rozdíl od množiny, může obsahovat stejný prvek vícekrát a je uspořádaný. Příkladem n-tice je například dvojice (a, b).

2.2 Relace a funkce

Definice 2.1. Nechť M_1 a M_2 jsou dvě množiny. *Kartézským součinem* M_1 a M_2, který zapisujeme $M_1 \times M_2$, je množina dvojic definovaná jako

$$M_2 \times M_2 = \{(x_1, x_2) \mid x_1 \in M_1 \text{ a } x_2 \in M_2\}$$

Definice 2.2. Nechť M_1 a M_2 jsou dvě množiny. *Binární relace* ρ z množiny M_1 do M_2, je libovolnou podmnožinou jejich kartézského součinu, tedy

$$\rho \subseteq M_1 \times M_2$$

Definice 2.3. Nechť je ψ relací z množiny M_1 do M_2. Pokud platí, že pro každé $x \in M_1$

$$card\,(\{y \mid y \in M_2 \text{ a } (x, y) \in \psi\}) \leq 1$$

pak říkáme, že ψ je *funkcí* z množiny M_1 do M_2.

2.3 Řetězce a jazyky

Definice 2.4. *Abeceda* je konečná neprázdná množina prvků, které nazýváme *symboly.*

Definice 2.5. Nechť Σ je abeceda, potom

- prázdný řetězec ε je řetězcem nad abecedou Σ
- pokud je x řetězcem nad abecedou Σ a $a \in \Sigma$, potom xa je řetězcem nad Σ

Definice 2.6. Nechť x je řetězcem nad abecedou Σ. Poté *délku řetězce*, kterou značíme $|x|$, definujeme jako

- pokud $x = \varepsilon$, pak $|x| = 0$
- pokud $x = a_1 \ldots a_n$, pak $|x| = n$, pro $n \geq 1$ a $a_i \in \Sigma$ pro všechna $i = 1, \ldots, n$

Definice 2.7. Nechť x a y jsou řetězce nad abecedou Σ. Výsledkem *konkatenace*, nebo také *zřetězení*, x a y je nový řetězec xy.

Definice 2.8. Nechť x je řetězcem nad abecedou Σ. Pak *n-tou mocninu řetězce* x pro $n \geq 0$ definujeme jako

- $x^0 = \varepsilon$
- $x^n = xx^{n-1}$, pro $n \geq 1$

Definice 2.9. Nechť x a y jsou řetězci nad abecedou Σ. Pokud existují takové řetězce z, z' nad abecedou Σ takové, že platí $zxz' = y$, potom je x *podřetězcem* y.

Definice 2.10. Nechť Σ^* označuje množinu všech řetězců, včetně ε, nad abecedou Σ. Každá podmnožina $L \subseteq \Sigma^*$ je *jazyk* nad abecedou Σ. Množinu všech neprázdných řetězců nad abecedou Σ značíme Σ^+ a platí, že

$$\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$$

2.4 Gramatiky

Gramatiky jsou speciální struktury sloužící pro generování jazyků. Můžeme o nich mluvit jako o *metajazycích*. Jsou tvořeny množinou pravidel, pomocí kterých je možné vygenerovat nějaký řetězec daného jazyka. Pokud lze každý řetězec vygenerovat právě jednou sekvencí použitých pravidel, mluvíme o takzvané *jednoznačné gramatice*. V opačném případě jde o *nejednoznačnou* gramatiku.

Definice 2.11. *Neomezená gramatika* je čtveřice

$$G = (N, T, P, S)$$

kde

- N je abeceda *neterminálů*,
- T je abeceda *terminálů* taková, že $N \cap T = \emptyset$,
- P je konečná relace z $(N \cup T)^* N (N \cup T)^*$ do $(N \cup T)^*$
- S je *počáteční symbol*, $S \in N$.

Jednotlivá pravidla, zapisovaná jako $r = (u, v) \in P$, nazýváme *přepisovací pravidla*, nebo také *derivační pravidla*. Častěji také používáme formu zápisu $r : u \to v$, popřípadě pouze $u \to v$.

2.4.1 Chomského hierarchie

Noam Chomsky v [2] zavedl čtyřstupňovou hierarchii gramatik, kde se každá úroveň liší svojí vyjadřovací silou. Gramatiky, které se nacházejí v hierarchii níže, tvoří podmnožinu jim nadřazených gramatik. Tento vztah je ilustrován obrázkem 2.1.

Definice 2.12. *Rekurzivně spočetné*, nebo také gramatiky **typu 0**, jsou všechny neomezené gramatiky. Jazyky generované rekurzivně spočetnými gramatikami se nazývají *rekurzivně spočetné* a zkráceně se značí **RE**.

Definice 2.13. *Kontextově senzitivní*, nebo také gramatika **typu 1**, je taková neomezená gramatika

$$G = (N, T, P, S)$$

pro kterou platí, že všechna její pravidla $u \to v$ z P mají formát

$$u = x_1 A x_2,\ v = x_1 y x_2$$

kde $x_1, x_2 \in (N \cup T)^*$, $A \in N$ a $y \in (N \cup T)^+$. Jazyky generované kontextově senzitivními gramatikami se nazývají *kontextově senzitivní* a zkráceně se značí **CS**.

Definice 2.14. *Bezkontextová*, nebo také gramatika **typu 2**, je taková neomezená gramatika

$$G = (N, T, P, S)$$

pro kterou platí, že všechna její pravidla $u \to v$ z P mají formát

$$A \to x$$

kde $A \in N$ a $x \in (N \cup T)^*$. Jyzyky generované bezkontextovými gramatikami se nazývají *bezkontextové* a zkráceně se značí **CF**.

Definice 2.15. *Regulární*, nebo také gramatika **typu 3**, je taková neomezená gramatika

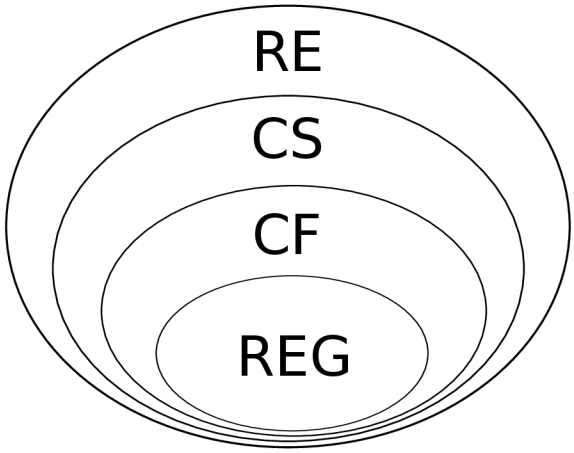
$$G = (N, T, P, S)$$

pro kterou platí, že všechna její pravidla $u \to v$ z P mají formát

$$A \to aB \text{ nebo } A \to a$$

kde $A, B \in N$ a $a \in T$. Jyzyky generované regulárními gramatikami se nazývají *regulární* a zkráceně se značí **REG**.

Věta 2.1. **REG** $\subset$ **CF** $\subset$ **CS** $\subset$ **RE**. Pro více informací viz [2].



Obrázek 2.1: Chomského hierarchie [2]

2.5 Automaty

Automat je zařízení schopné poznat řetězce dané gramatiky. Zatímco gramatiky slouží pro vytváření řetězců, automaty pracují opačně a pomocí analýzy vstupního řetězce rozhodnou, zda náleží příslušnému jazyku. V této sekci budou definováni dva základní představitelé automatů — *konečný automat*, schopný rozpoznat regulární jazyky, a *zásobníkový automat*, který je schopen rozpoznat jazyky bezkontextové.

Definice 2.16. *Konečný automat* je pětice

$$A = (Q, \Sigma, \delta, q_0, F)$$

kde

- Q je konečná množina *stavů*,
- Σ je *vstupní abeceda*,
- δ je *přechodová funkce* ve tvaru $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to Q$,
- q_0 je *počáteční stav*, $q_0 \in Q$,

- F je množina *konečných stavů*, $F \subseteq Q$.

Definice 2.17. Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je konečný automat. *Konfigurace* A je dvojice $(s, x) \in Q \times \Sigma^*$, kde

- s je aktuální stav a
- x je doposud nepřečtená část vstupního řetězce.

Definice 2.18. Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je konečný automat. *Přechod* A, značený $\vdash$, je definován jako

$$(s, x) \vdash (q, y)$$

pouze pokud $x = ay$ a $q \in \delta(s, a)$, kde $s, q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $x, y \in \Sigma^*$. Zápis $\vdash^n$, $n \geq 0$, značí posloupnost n přechodů. Pokud není n shora omezené, požíváme označení $\vdash^*$. Je-li $n > 0$, píšeme $\vdash^+$.

Definice 2.19. Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je konečný automat. Jazyk přijímaný A se značí $L(A)$ a je definován jako

$$L(A) = \{x \in \Sigma^* \mid (q_0, x) \vdash^* (f, \varepsilon), f \in F\}$$

Definice 2.20. *Zásobníkový automat* je sedmice

$$A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

kde

- Q je konečná množina *stavů*,
- Σ je *vstupní abeceda*,
- Γ je *zásobníková abeceda*,
- δ je *přechodová funkce* ve tvaru $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$,
- q_0 je *počáteční stav*, $q_0 \in Q$,
- Z_0 je *počáteční zásobníkový symbol*, $Z_0 \in \Gamma$,
- F je množina *konečných stavů*, $F \subseteq Q$.

Definice 2.21. Nechť $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je zásobníkový automat. *Konfigurace* A je trojice $(s, x, \alpha) \in Q \times \Sigma^* \times \Gamma^*$, kde

- s je aktuální stav,
- x je doposud nepřečtená část vstupního řetězce a
- α je obsah zásobníku; nejlevější symbol je jeho vrcholem.

Definice 2.22. Nechť $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je zásobníkový automat. *Přechod*, nebo také *krok*, A, značený $\vdash$, je definován jako

$$(s, x, \alpha) \vdash (q, y, \alpha')$$

pouze pokud $x = ay$, $\alpha = Z\beta$, $\alpha' = \gamma\alpha$ a $(q, \gamma) \in \delta(s, a, Z)$ kde $s, q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $x, y \in \Sigma^*$, $Z \in \Gamma$ a $\alpha, \alpha', \beta, \gamma \in \Gamma'$. Zápis $\vdash^n$, $n \geq 0$, značí posloupnost n přechodů. Pokud není n shora omezené, požíváme označení $\vdash^*$. Je-li $n > 0$, píšeme $\vdash^+$.

Pro zásobníkové automaty existují tři způsoby, jakými mohou přijímat jazyk. Jedná se o přijetí

- dosažením koncového stavu,
- vyprázdněním zásobníku,
- dosažením koncového stavu a vyprázdněním zásobníku.

Definice 2.23. Nechť $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je zásobníkový automat. Jazyk přijímaný A dosažením koncového stavu se značí $L_f(A)$ a je definován jako

$$L_f(A) = \{x \in \Sigma^* \mid (q_0, x, Z_0) \vdash^* (f, \varepsilon, \alpha), f \in F, \alpha \in \Gamma^*\}$$

Jazyk přijímaný A vyprázdněním zásobníku se značí $L_e(A)$ a je definován jako

$$L_e(A) = \{x \in \Sigma^* \mid (q_0, x, Z_0) \vdash^* (q, \varepsilon, \varepsilon), q \in Q\}$$

Jazyk přijímaný A dosažením koncového stavu a vyprázdněním zásobníku se značí $L_{fe}(A)$ a je definován jako

$$L_{fe}(A) = \{x \in \Sigma^* \mid (q_0, x, Z_0) \vdash^* (f, \varepsilon, \varepsilon), f \in F\}$$

Kapitola 3

PC gramatické systémy

Tyto systémy byly poprvé zavedeny v [10]. Princip funkce těchto systémů tkví v paralelní práci jejich komponent, kdy každá generuje vlastní větnou formu. Pokud je některým pravidlem zaveden do větné formy speciální, takzvaný dotazovací, symbol, systém provede komunikační krok. Tato komunikace vede k tomu, že dojde k nahrazení dotazovacího symbolu větnou formou komponenty, na kterou se tento symbol odkazuje. Jazyk generovaný systémem jako celkem je poté jazyk generovaný jednou určenou komponentou, zpravidla se jedná o první komponentu. Existují různé varianty těchto systémů lišících se zejména chováním svých komponent po komunikačním kroku, omezeními, které komponenty mohou iniciovat komunikaci a způsoby, jakými se zahajuje a probíhá komunikace. Několik základních variant si představíme dále. PC gramatické systémy také nekladou žádné omezení na to, které druhy gramatik mohou být jejich komponentami. V této práci budeme nicméně uvažovat pouze ty systémy, jejichž komponenty jsou nanejvýš bezkontextové (2.14).

Definice 3.1. *PC gramatický systém* [4] stupně n, zkráceně $PCGS(n)$, $n \geq 1$, je $(n+3)$-tice

$$\Gamma = (N, K, T, G_1, \ldots, G_n)$$

kde

- N je abeceda *neterminálů*,
- $K = \{K_1, \ldots, K_n\}$ je abeceda *dotazovacích symbolů*, kde K_i odkazuje na G_i, $1 \leq i \leq n$,
- T je abeceda *terminálů*,
- gramatika $G_i = (N \cup K, T, P_i, S_i)$, $1 \leq i \leq n$, je *komponenta* Γ

a platí, že $N \cap K \cap T = \emptyset$. G_1 označujeme za *master* komponentu.

Definice 3.2. Nechť $\Gamma = (N, K, T, G_1, \ldots, G_n)$ je $PCGS(n)$. Říkáme, že Γ je *centralizovaný PC gramatický systém*, zkráceně $CPCGS(n)$, pokud pouze master komponenta může generovat požadavky na komunikaci. Tedy G_1 jako jediná může obsahovat pravidla ve tvaru

$$A \to x, A \in N, x \in (N \cup K \cup T)^*$$

a všechny ostatní G_i, $2 \leq i \leq n$ mohou obsahovat pouze pravidla ve tvaru

$$A \to x, A \in N, x \in (N \cup T)^*$$

Definice 3.3. Nechť $\Gamma = (N, K, T, G_1, \ldots, G_n)$ je $PCGS(n)$. Říkáme, že Γ je *navracející se PC gramatický systém*, zkráceně $RPCGS(n)$, pokud po ukončení komunikačního kroku, ve kterém komponenta G_i, $1 \leq i \leq n$ komunikovala svoji větnou formu $x_i \in (N \cup T)^*$, je $x_i = S_i$.

Definice 3.4. Nechť $\Gamma = (N, K, T, G_1, \ldots, G_n)$ je $PCGS(n)$. *Konfigurace* [4] Γ je n-tice

$$(x_1, \ldots, x_n)$$

kde $x_i \in (N \cup K \cup T)^*$, $1 \leq i \leq n$, je aktuální větná forma komponenty G_i.

Definice 3.5. Nechť $\Gamma = (N, K, T, G_1, \ldots, G_n)$ je $PCGS(n)$. *Derivačním krokem* [4] Γ, značeným symbolem $\Rightarrow$, je

$$(x_1, \ldots, x_n) \Rightarrow (y_1, \ldots, y_n)$$

pokud platí jedna z následujících podmínek

1. Pro všechny x_i, $1 \leq i \leq n$ platí, že $|x_i|_K = 0$. Potom pro všechna i, $1 \leq i \leq n$, $x_i \Rightarrow_{G_i} y_i$ pokud $x_i \notin T^*$. Pro taková x_i kde $x_i \in T^*$ je $y_i = x_i$.

2. Pro všechna x_i, $1 \leq i \leq n$, kde $|x_i|_K \neq 0$ píšeme $x_i = z_1 Q_{i_1} z_2 Q_{i_2} \ldots z_t Q_{i_t} z_{t+1}$, kde $z_j \in (N \cup T)^*$, $1 \leq j \leq t+1$ a $Q_{i_l} \in K$, $1 \leq l \leq t$. Potom $y_i = z_1 u_{i_1} z_2 u_{i_2} \ldots z_t u_{i_t} z_{t+1}$, kde $u_{i_l} = x_{i_l}$ pokud $|x_{i_l}|_K = 0$ a $u_{i_l} = Q_{i_l}$ pokud $|x_{i_l}|_K \neq 0$, $1 \leq l \leq t$. Pokud $u_{i_l} = x_{i_l}$ tak v navracejících se systémech $y_{i_l} = S_{i_l}$, v nenavracejících se systémech $y_{i_l} = x_{i_l}$, $1 \leq l \leq t$. Pro všechna j, $1 \leq j \leq n$, pro která y_j není definována, platí $y_j = x_j$.

Bod jedna definice 3.5 zavádí takzvaný *derivační krok*, při kterém žádná komponenta systému nevyžaduje komunikaci, a tak všechny provedou přesně jeden derivační krok podle svých pravidel. Takové $PCGS(n)$, jejichž komponenty provádí současně vždy jednu derivaci, říkáme *synchronizované*. Sytémům, ve kterých tato podmínka neplatí, říkáme *nesynchronizované* [9], a v této práci se jim nebudeme věnovat. Pokud nelze aplikovat žádné pravidlo, a stávající větná forma není složena pouze z terminálních symbolů, dojde k zablokování derivace.

Druhý bod definice 3.5 zavádí *komunikační krok*. Komunikační krok má vždy přednost před krokem derivačním. Během tohoto kroku dojde k nahrazení dotazovacího symbolu větnou formou komponenty, na kterou se odkazuje, za podmínky, že tato větná forma neobsahuje žádný dotazovací symbol. V jednom komunikačním kroku jsou takovou větnou formou nahrazeny všechny příslušné dotazovací symboly ve všech komponentách [4]. Existují také další definice komunikačního kroku, pro další informace odkazujeme čtenáře na [4]. Během komunikačního kroku může také dojít k zablokování systému. Zablokování nastane, pokud vznikne kruhová reference — v systému není žádná komponenta, na kterou směřuje dotaz, jejíž větná forma neobsahuje žádný dotazovací symbol.

Definice 3.6. Nechť $\Gamma = (N, K, T, G_1, \ldots, G_n)$ je $PCGS(n)$. Jazyk generovaný Γ je

$$L(\Gamma) = \{x \in T^* \mid (S_1, S_2, \ldots, S_n) \Rightarrow^* (x, \gamma_2, \ldots, \gamma_n), \gamma_i \in (N \cup K \cup T)^*, 2 \leq i \leq n\}$$

Z definice je patrné, že jazyk generovaný celým systémem, odpovídá jazyku generovanému jeho master komponentou. V momentě, kdy master komponenta vygeneruje řetězec složený pouze z terminálních symbolů, dojde k zastavení celého systému. V takovémto případě jsou aktuální větné formy $\gamma_2, \ldots, \gamma_n$ ignorovány.

Příklad 3.1. Na závěr této kapitoly uvedeme příklad jednoho $PCGS(n)$.

$$\Gamma = (\{S_1, S_2\}, \{K_1, K_2\}, \{a, b\}, (N \cup K, T, P_1, S_1), (N \cup K, T, P_2, S_2))$$

$$\begin{aligned} P_1 = \{&S_1 \to S_1 \\ &S_1 \to K_1 K_2\} \end{aligned} \qquad \begin{aligned} P_2 = \{&S_2 \to aS_2 \\ &S_2 \to bS_2 \\ &S_2 \to a \\ &S_2 \to b\} \end{aligned}$$

Jazyk generovaný tímto systémem je

$$L(\Gamma) = \{xx \mid x \in \{a, b\}^+\}$$

Druhá komponenta tohoto systému generuje řetězec náhodně složený ze symbolů a a b. V určitém okamžiku poté provede druhá komponenta nedeterministicky takovou derivaci, která způsobí, že její větná forma bude obsahovat pouze terminály. Do tohoto okamžiku provadí první komponenta derivaci $S_1 \to S_1$, tedy efektivně nedělá nic. Ve stejný okamžik, kdy druhá komponenta dogeneruje řetězec terminálů, použije pravidlo $S_1 \to K_1 K_2$. Tím dojde k nahrazení K_2 již zmíněným řetězcem terminálů, který je následně zdvojen tím, že komponenta komunikuje sama se sebou.

Z tohoto příkladu je patrné, že kombinací relativně jednoduchých částí, komponent, lze vytvořit systém schopný generovat mnohem komplikovanější řetězce, než jakých jsou schopny samotné jeho komponenty. Za tuto sílu je rovným dílem zodpovědná komunikace mezi jednotlivými komponentami a také to, že všechny komponenty pracují synchronně.

K předchozímu příkladu si také uveďme, že se jedná o centralizovaný systém, protože pouze první, master, komponenta, obsahuje pravidla generující dotazovací symboly. Další zajímavou vlastnosti tohoto systému je to, že jazyk jím generovaný je stejný, ať už ho budeme uvažovat jako navracející se nebo ne. To je dáno tím, že ke komunikaci mezi komponentami může dojít pouze jednou, a tedy stav větné formy druhé komponenty se nemůže po komunikaci projevit na výsledném řetězci.

Kapitola 4

PC systémy zásobníkových automatů

V této kapitole si představíme paralelně komunikující systémy zásobníkových automatů. Jako předchůdce těchto systémů byly v [6] představeny systémy složené z konečných automatů, které si navzájem komunikovaly stavy. Na tuto práci bylo později navázáno v [3] uvedením systémů, jejichž komponenty byly zásobníkové automaty komunikující obsahy svých zásobníků. Protože činnost konečného automatu můžeme simulovat automatem zásobníkovým, budeme se v této práci věnovat právě PC systémům zásobníkových automatů.

Takové systémy, které zkráceně označujeme jako PCPA, do značné míry kopírují chování PC gramatických systémů, zejména co se týče komunikace mezi komponentami. Ke komunikaci mezi komponentami dochází tehdy, objeví-li se na vrcholu zásobníku nějaké komponenty dotazovací symbol. Takový symbol je nahrazen obsahem zásobníku té komponenty, na kterou se odkazuje. Zde je rozdíl oproti PC gramatickým systémům, kde ke komunikaci došlo, kdykoliv se dotazovací symbol objevil na libovolné pozici větné formy. U PC systémů zásobníkových automatů tak může dojít k naplánování komunikace v budoucnu — uložením jednoho či více dotazovacích symbolů na zásobník pod jiný, běžný, zásobníkový symbol. Opět také můžeme tyto systémy členit na centralizované a necentralizované, a na navracející se a nenavracející se.

Definice 4.1. *PC systém zásobníkových automatů* stupně n, zkráceně $PCPA(n)$, $n \geq 1$, je $(n+3)$-tice

$$\mathcal{A} = (\Sigma, \Gamma, K, A_1, \ldots, A_n)$$

kde

- Σ je *vstupní abeceda*,
- Γ je *zásobníková abeceda*,
- $K = \{K_1, \ldots, K_n\}$ je abeceda *dotazovacích symbolů*, kde K_i odkazuje na A_i, $1 \leq i \leq n$,
- zásobníkový automat $A_i = (Q_i, \Sigma, \Gamma \cup K, \delta_i, q_i, Z_i, F_i)$, $1 \leq i \leq n$, je *komponenta* $\mathcal{A}$

a platí, že $\Sigma \cap \Gamma \cap K = \emptyset$. A_1 označujeme za *master* komponentu.

Definice 4.2. Nechť $\mathcal{A} = (\Sigma, \Gamma, K, A_1, \ldots, A_n)$ je $PCPA(n)$. *Konfigurací* $\mathcal{A}$ je $3n$-tice

$$(s_1, x_1, \alpha_1, \ldots, s_n, x_n, \alpha_n)$$

kde pro všechna $1 \leq i \leq n$

- $s_i \in Q_i$ je aktuální stav komponenty A_i,
- $x_i \in \Sigma^*$ je doposud nepřečtená část vstupního řetězce komponenty A_i,
- $\alpha_i \in (\Gamma \cup K)^*$ je obsah zásobníku komponenty A_i; nejlevější symbol je jeho vrcholem

Definice 4.3. Nechť $\mathcal{A} = (\Sigma, \Gamma, K, A_1, \ldots, A_n)$ je $PCPA(n)$. Říkáme, že $\mathcal{A}$ je *centralizovaný PC systém zásobníkových automatů*, zkráceně $CPCPA(n)$, pokud pro přechody ve tvaru

$$\delta_i(s_i, a_i, \alpha_i) = \{(q_i, \beta_i)\},$$

kde $1 \leq i \leq n$, $s_i, q_i \in Q_i$, $a_i \in \Sigma$, $\alpha_i \in \Gamma$, platí, že $\beta_i \in (\Gamma \cup K)^*$ pouze pro master komponentu A_1, a pro všechna ostatní A_j, $2 \leq j \leq n$ je $\beta_j \in \Gamma^*$.

Definice 4.4. Nechť $\mathcal{A} = (\Sigma, \Gamma, K, A_1, \ldots, A_n)$ je $PCPA(n)$. Říkáme, že $\mathcal{A}$ je *navracející se PC systém zásobníkových automatů*, zkráceně $RPCPA(n)$, pokud po ukončení komunikačního kroku, ve kterém byl proveden dotaz na komponentu A_i, $1 \leq i \leq n$, je obsah jejího zásobníku zredukován na Z_i. K této redukci dochází teprve tehdy, až byl obsah zásobníku předán všem žádajícím komponentám.

Definice 4.5. Nechť $\mathcal{A} = (\Sigma, \Gamma, K, A_1, \ldots, A_n)$ je $PCPA(n)$. *Přechodem* $\mathcal{A}$, označujeme jej symbolem $\vdash$, je

$$(s_1, x_1, B_1\alpha_1, \ldots, s_n, x_n, B_n\alpha_n) \vdash (p_1, y_i, \beta_1, \ldots, p_n, y_n, \beta_n),$$

kde $s_i, p_i \in Q_i$, $x_i, y_i \in \Sigma^*$, $B_i \in \Gamma$, $\alpha_i, \beta_i \in \Gamma^*$, $1 \leq i \leq n$, pokud platí právě jedna z následujících podmínek

1. Pro všechna i, $1 \leq i \leq n$, platí, že $B_i \notin K$ a $x_i = a_i y_i$, $a_i \in \Sigma \cup \{\varepsilon\}$, $(p_i, \beta_i') \in \delta_i(s_i, a_i, B_i)$, $\beta_i = \beta_i'\alpha_i$
2. Pro všechna taková i, $1 \leq i \leq n$, kde $B_i = K_j$ a $B_j \notin K$, $1 \leq j \leq n$, $\beta_i = B_j\alpha_j\alpha_i$. Pokud je $\mathcal{A}$ navracející se systém, platí navíc, že $\beta_j = Z_j$. Pro všechna ostatní r, $1 \leq r \leq n$ je $\beta_r = B_r\alpha_r$ a pro všechna t, $1 \leq t \leq n$, je $y_t = x_t$, $p_t = s_t$.

Podobně jako u PC gramatických systémů zavádí předchozí definice dva druhy kroků, které může PCPA uskutečnit. První se provede tehdy, není-li na vrcholu zásobníku žádné komponenty dotazovací symbol. V tomto případě provede každá komponenta přechod do nové konfigurace podle své přechodové funkce. Nemůže-li nějaká komponenta provést žádný přechod, dochází k zablokování systému. Stejně jako jsme si uvedli u PC gramatických systémů, dochází právě k jednomu současnému přechodu všech komponent — jde o *synchronizované* systémy. Jejich *nesynchronizovaným* sourozencům se v tomto textu nebudeme věnovat.

V případě, že se na vrcholu zásobníku alespoň jedné komponenty nachází dotazovací symbol, proběhne komunikační krok. Každý takový symbol je nahrazen obsahem zásobníku té komponenty, na kterou se odkazuje. Zde je vhodné uvést, že vrchol zásobníku, který je komunikován, bude tvořit i vrchol cílového zásobníku. Také platí, že dojde-li během jednoho komunikačního kroku v navracejícím se systému k více dotazům na jednu komponentu, je obsah jejího zásobníku nejprve odeslán všem žádajícím komponentám a teprve poté dojde k jeho redukci na počáteční zásobníkový symbol. Opět také existuje možnost uváznutí systému pokud dojde ke kruhovému dotazu.

Definice 4.6. Nechť $\mathcal{A} = (\Sigma, \Gamma, K, A_1, \ldots, A_n)$ je $PCPA(n)$. Jazyk přijímaný $\mathcal{A}$ dosažením konečného stavu všemi komponentami značíme $L_f(\mathcal{A})$ a definujeme jej jako

$$L_f(\mathcal{A}) = \{x \in \Sigma^* \mid (q_1, x, Z_1, \ldots, q_n, x, Z_n) \vdash^* (s_1, \varepsilon, \alpha_1, \ldots, s_n, \varepsilon, \alpha_n), s_i \in F_i, 1 \leq i \leq n\}$$

Jazyk přijímaný $\mathcal{A}$ vyprázdněním zásobníků všech jeho komponent značíme $L_e(\mathcal{A})$ a definujeme jej jako

$$L_e(\mathcal{A}) = \{x \in \Sigma^* \mid (q_1, x, Z_1, \ldots, q_n, x, Z_n) \vdash^* (s_1, \varepsilon, \varepsilon, \ldots, s_n, \varepsilon, \varepsilon), 1 \leq i \leq n\}$$

Jazyk přijímaný $\mathcal{A}$ kombinací obou výše uvedených variant značíme $L_{fe}(\mathcal{A})$ a definujeme jej jako

$$L_{fe}(\mathcal{A}) = \{x \in \Sigma^* \mid (q_1, x, Z_1, \ldots, q_n, x, Z_n) \vdash^* (s_1, \varepsilon, \varepsilon, \ldots, s_n, \varepsilon, \varepsilon), s_i \in F_i, 1 \leq i \leq n\}$$

Příklad 4.1. Zakončeme tuto kapitolu příkladem jednoho PC systému zásobníkových automatů, konkrétně $CPCPA(2)$. Pro zachování přehlednosti si jej nadefinujeme primárně pomocí přechodových funkcí jeho komponent.

$$\begin{aligned}
\delta_1(q_1, X, Z_1) &= \{(q_1, Z_1)\}, & \delta_2(q_2, X, Z_2) &= \{(q_2, XZ_2)\},\\
\delta_1(q_1, c, Z_1) &= \{(s_1, K_2Z_1)\}, & \delta_2(q_2, X, Y) &= \{(q_2, XY)\},\\
\delta_1(s_1, \varepsilon, X) &= \{(s_1, K_2X)\}, & \delta_2(q_2, c, X) &= \{(s_2, X)\},\\
\delta_1(q_1, \varepsilon, Z_2) &= \{(p_1, \varepsilon)\}, & \delta_2(s_2, \varepsilon, X) &= \{(s_2, \varepsilon)\},\\
\delta_1(p_1, X, X) &= \{(p_2, \varepsilon)\}, & \delta_2(s_2, \varepsilon, Z_2) &= \{(s_f, Z_2)\},\\
\delta_1(p_2, \varepsilon, X) &= \{(p_2, \varepsilon)\}, & \delta_2(s_f, \varepsilon, Z_2) &= \{(s_f, Z_2)\},\\
\delta_1(p_2, \varepsilon, Z_2) &= \{(p_1, \varepsilon)\}, & \delta_2(s_f, X, Z_2) &= \{(s_f, XZ_2)\}\\
\delta_1(p_1, \varepsilon, Z_1) &= \{(p_f, Z_1)\}
\end{aligned}$$

kde $X, Y \in \{a, b\}$, a množiny konečných stavů jednotlivých komponent jsou $F_1 = \{p_f\}$ a $F_2 = \{s_f\}$. Jazyk, který tento systém přijímá je

$$L(\mathcal{A}) = \{xcx \mid x \in \{a, b\}^+\}$$

Obě komponenty tohoto systému čtou vstupní řetězec, přičemž první komponenta si jednotlivé symboly ukládá na svůj zásobník. Jakmile master komponenta narazí na symbol c, vyžádá si obsah zásobníku druhé komponenty, která momentálně obsahuje reverzovaný řetězec x, označme si jej $reversal(x)$. Nyní v každém dalším kroku odstraní druhá komponenta jeden symbol z vrcholu svého zásobníku a první komponenta si vyžádá jeho obsah. Tento proces se opakuje tak dlouho, dokud zásobník druhé komponenty není tvořen pouze symbolem Z_2. V tento okamžik se na zásobníku nachází takovýto řetězec

$$Z_2 reversal(x^{(1)}) Z_2 reversal(x^{(2)}) Z_2 \ldots Z_2 reversal(x^{(|x|-1)}) Z_2 reversal(x) Z_2$$

kde $x^{(l)}$ označujeme takový řetězec, který je tvořen l prvními znaky x. Tomuto řetězci také říkáme *prefix* x délky l. Nyní již jen první komponenta postupně porovnává symbol, který je na zásobníku uložen bezprostředně po Z_2 se symbolem ze vstupní pásky.

4.1 Nedeterminismus

V této sekci si uvedeme zajímavý efekt nedeterminismu v jednotlivých komponentách na celý systém. Pokud uvažujeme samostatný zásobníkový automat, nebo také $PCPA(1)$, je zřejmé, že pokud může takový automat provést přechod z jedné konfigurace do n možných, znamená to také, pokud budeme předpokládat, že nebyly použity žádné heuristiky, že existuje n větví stavového prostoru, kterými může automat dále pokračovat. Pokud ale taková situace nastane v PCPA, počet možných konfigurací, do kterých může systém jako celek přejít, je roven počtu kombinací mezi konfiguracemi, do kterých mohou přejít jeho komponenty. Uvažujme konfiguraci $(s_1, x_1, \alpha_1, \ldots, s_n, x_n, \alpha_n)$ nějakého $PCPA(n)$. Počet konfigurací, do kterých může systém jako celek přejít, je roven

$$\prod_{i=1}^{n} card\left(\delta_i\left(s_i, x_i, \alpha_i\right) \cup \delta_i\left(s_i, \varepsilon, \alpha_i\right)\right)$$

a samotná množina možných konfigurací je dána

$$\{\delta_1(s_1, x_1, \alpha_1) \cup \delta_1(s_1, \varepsilon, \alpha_1)\} \times \cdots \times \{\delta_n(s_n, x_n, \alpha_n) \cup \delta_n(s_n, \varepsilon, \alpha_n)\}$$

Tento vztah ukazuje, že i malé množství nedeterminismu v komponentách může mít zásadní dopad na celý systém. Později si v této práci představíme jednu novou vlastnost pro komponenty PCPA, která v nich umožní alespoň částečně omezit množství nedeterminismu.

Na tento problém ještě navážeme dalším, který s ním souvisí. Synchronizovaný systém může provést přechod do nové konfigurace právě tehdy, může-li každá jeho komponenta provést alespoň jeden přechod. Pro čtenáře jistě není težké představit si, že jen zřídka kdy budou všechny komponenty systému potřebovat pro splnění své logické funkce (analýzy určitých podřetězců vstupního řetězce) stejný počet přechodů. Daleko pravděpodobnější je, že každá komponenta dokončí svoji práci, například dosažením koncového stavu, po různém počtu přechodů. V tuto chvíli je ale třeba zajistit, aby tato komponenta již v takovémto stavu setrvala a přitom byla nadále schopna provést vždy alespoň jeden přechod, tak aby nedošlo k nechtěnému zablokování celého systému. Tohoto lze například dosáhnout takovým přechodem, který nezmění ani stav ani obsah zásobníku příslušné komponenty. Navíc je dále nutné zajistit, aby každá komponenta přečetla celý vstupní řetězec, a to včetně těch částí, zejména koncových, které nejsou pro její funkci vůbec potřebné.

Pokud se vrátíme k příkladu 4.1, vidíme že u druhé komponenty, která svoji úlohu dokončí dříve než první, je tohoto dosaženo hodnotami přechodové funkce $\delta_2(s_f, X, Z_2) = \{(s_f, Z_2)\}$ a $\delta_2(s_f, \varepsilon, Z_2) = \{(s_f, Z_2)\}$. Po dosažení koncového stavu, jehož dosažení je bodem kdy komponenta splnila svoji funkci, využije první přechod aby dočetla zbývající část vstupního řetězce, a poté pomocí ε-přehodu setrvá v koncovém stavu — tento přechod lze totiž provést vždy.

Přestože se jedná o řešení, které elegantně adresuje výše popsaný problém, pojí se s ním dvojice problémů. Prvním a méně závažným je to, že bylo nutné explicitně uvést tyto přechody, které nesouvisejí s hlavní funkcí této komponenty. Druhým, a nyní již zásadním, je to, že pokud se na tyto dva přechody podíváme zblízka, vidíme, že dokud nebyl zcela přečten vstupní řetězec, lze provést oba dva zároveň. Není také těžké představit si, že takto dojde k postupnému generování ohromného množství stavů, do kterých může systém přejít.

Tento problém by bylo možné vyřešit zavedením speciálního symbolu, který by označoval konec vstupního řetězce, pomocí něhož by se provedl přechod do stavu, ve kterém by komponenta setrvala již bez potřeby nedeterminismu. To by ale ještě více zkomplikovalo návrh jednotlivých komponent.

Kapitola 5

Modifikace PC systémů zásobníkových automatů

V této kapitole si představíme takovou modifikaci pro komponenty PC systémů zásobníkových automatů, která umožní omezit dopady problémů popsaných v 4.1. Tato modifikace zároveň umožní mírně zjednodušit návrh PC systémů zásobníkových automatů tím, že odstraňuje potřebu v komponentách explicitně definovat některé přechody zavedením nového druhu přechodu — takzvaného *implicitního*.

Motivací pro zavedení této modifikace je odstranění potřeby navrhovat pro komponenty ty přechody, které jí umožňují korektně fungovat i poté, co již byla ukončena její logická funkce. Jedná se tedy hlavně o přechody, které zajistí, že komponenta po přečtení celého vstupního řetězce setrvá v koncovém stavu.

Definice 5.1. Nechť $\mathcal{A} = (\Sigma, \Gamma, K, A_1, \ldots, A_n)$ je $PCPA(n)$ a $A_i = (Q_i, \Sigma, \Gamma \cup K, \delta_i, q_i, Z_i, F_i)$, $1 \leq i \leq n$, je komponenta $\mathcal{A}$. Pro libovolnou konfiguraci (s_i, x_i, α_i), $s_i \in Q_i$, $x_i \in \Sigma^*$, $\alpha_i \in \Gamma^*$, zásobníkového automatu A_i, definujeme *implicitní přechod* jako

$$\delta_i(s_i, a_i, \beta_i) = \{(s_i, \alpha_i)\}$$

kde $x_i = a_i y_i$, $\alpha_i = \beta_i \gamma_i$, pouze pokud platí všechny následující podmínky

- $\beta_i \notin K$,
- $\delta(s_i, a_i, \beta_i) = \emptyset$,
- (s_i, x_i, α_i) je taková konfigurace, ve které automat přijímá vstupní řetězec.

Tyto implicitní hodnoty přechodové funkce δ umožňují komponentám PC systémů zásobníkových automatů setrvat v konfiguraci, ve které komponenta přijala vstupní řetězec, bez potřeby explicitně specifikovat přechody, kterými by bylo dosaženo stejného výsledku.

Výhoda využití této modifikace leží ve dvou rovinách. První je již zmíněné částečné odstranění nutnosti zahrnout do návrhu takovou funkcionalitu, která přímo nesouvisí s primární úlohou dané komponenty. Druhá výhoda je, že tento přechod se chová jako přechod s nejnižší možnou prioritou — lze jej provést pouze tehdy, není-li možné provést jakýkoliv jiný přechod nebo komunikační krok. Tyto podmínky v kombinaci s podmínkou, že ve stávající konfiguraci již automat přijal vstupní řetězec, zajišťují aby nedošlo k nekorektní změně stavu komponenty, která by mohla mít vliv na zbytek systému.

Popišme si nyní jaký dopad by měla tato modifikace na systém z příkladu 4.1. Z tohoto systému bychom mohli odstranit hodnotu přechodové funkce $\delta_2(s_f, \varepsilon, Z_2) = \{(s_f, Z_2)\}$.

Problémy s ní spojené jsme si popsali v 4.1. Jejím odstraněním dojde ke zjednodušení, byť k malému, druhé komponenty a především k odstranění nedeterminismu, který vznikal kombinací s přechodem $\delta_2(s_f, X, Z_2) = \{(s_f, Z_2)\}$. Nyní pomocí tohoto přechodu dojde k přečtení zbytku vstupního řetězce a teprve po jeho přečtení může druhá komponenta využít implicitní přechod pro setrvání ve stavu s_f — bez zavedení nedeterminismu do systému.

Tento implicitní přechod, který jsme si výše definovali, není nepodobný instrukci **NOP** známé z jazyků symbolických instrukcí. Pokud je do systému zaveden, má za následek provedení přechodu, který žádným způsobem nezmění stav komponenty a tedy ani nemůže ovlivnit zbytek systému.

Kapitola 6

Modifikace PC gramatických systémů

V následujícím textu si představíme taková omezení pro PC gramatické systémy, která nám umožní jejich následný převod na PC systémy zásobníkových automatů. Samotný algoritmus pro převod představíme v následující kapitole. Zde si hned na začátek uveďme, že dále budeme uvažovat pouze systémy, jenž jsou synchronizované a jejichž komponenty při derivaci přepisují vždy nejlevější neterminální symbol ve své větné formě. V obecném PC gramatickém systému jeho komponenty synchronně generují řetězce symbolů, kterou jsou master komponentou skládány do jednoho výsledného řetězce. Naším cílem je tyto systémy omezit takovým způsobem tak, aby všechny jejich komponenty, vyjma té master, každou derivací generovaly právě jeden terminální symbol a maximálně jeden symbol neterminální. Navíc budeme dále vyžadovat, aby každá taková komponenta pracovala právě s jedním terminálním a jedním neterminálním symbolem. Tedy aby mohla generovat pouze řetězec libovolné délky právě jednoho symbolu. Master komponenta, na kterou se toto omezení nevztahuje, poté bude tvořit řídící jednotku, která podle určitého programu (daného jejími pravidly) bude skládat výsledný řetězec.

Myšlenkou stojící za tímto omezením je přiblížit takové gramatické systémy těm zásobníkovým. V tomto odstavci budeme komponentami myslet všechny ty, jenž nejsou master komponentou. Důvodem pro zavedení požadavku, aby komponenty generovaly pouze řetězec jednoho symbolu, je vypořádání se s problémem zásobníku — pokud takový řetězec ukládáme při čtení na zásobník, dojde k jeho reverzaci, tedy na vrcholu zásobníku je poslední přečtený symbol. Pokud by si takový zásobník vyžádala master komponenta, nemůže jej vždy porovnat se vstupem. Reverzace řetězce délky k znaku x, také x^k, je opět x^k a tedy pro tento případ zmíněný problém nenastane. Omezení na generování jednoho symbolu jednou derivací poté umožní zachovat synchronizaci během převodu — pokud gramatická komponenta vygeneruje řetězec během délky k během k kroků, může jej zásobníkový automat přečíst a uložit na svůj zásobník opět během k kroků.

Řetězec vytvořený takovýmto systémem je možné zpracovat PC systémem zásobníkových automatů tak, že každá jeho komponenta z něj extrahuje specifické části, které využívá master komponenta pro jeho analýzu. Ostatní části řetězce pak slouží každé komponentě jako zdroj synchronizace.

Následující sekce představí takový gramatický systém, jehož master komponenta je tvořena lineární gramatikou a ostatní komponenty jsou omezené regulární gramatiky.

6.1 Omezený lineární PC gramatický systém

Definice 6.1. *Omezený lineární PC gramatický systém* stupně n, je takový $PCGS(n)$, pro který platí následující podmínky

- Master komponenta G_1 je *lineární* gramatika. Lineární gramatika je taková bezkontextová gramatika (2.14), která na pravých stranách svých pravidel obsahuje nanejvýš jeden neterminální symbol.
- Všechny ostatní G_i, $2 \leq i \leq n$ jsou takové regulární gramatiky, které obsahují pouze pravidla ve tvaru $A \to xQB$, kde $A \in N_i$, $Q \in K \cup \{\varepsilon\}$, $B \in \{A, \varepsilon\}$, $x \in T_i$ a platí, že $\varepsilon \notin T_i$, $card(T_i) = 1$ a $card(N_i) = 1$. Těmto gramatikám se také říká *pravé regulární*.

Vraťmě se zde k příkladu 3.1. Je patrné, že tento systém nevyhovuje podmínkám výše uvedené definice, protože jeho druhá komponenta generuje řetězec složený ze dvou symbolů. Proto si zde uvedeme příklad takového systému, který jim již vyhovuje.

Příklad 6.1. Mějme $PCGS(3)$ $\Gamma = (\{S_1, S_1', S_2, S_3\}, K, \{a, b, c\}, G_1, G_2, G_3)$, kde

$$
\begin{aligned}
P_1 = \{ & S_1 \to abc, \\
& S_1 \to a^2b^2c^2, \\
& S_1 \to aS_1', \\
& S_1 \to a^3K_2, \\
& S_1' \to aS_1', \\
& S_1' \to a^3K_2, \\
& S_2 \to b^2K_2, \\
& S_3 \to c\}
\end{aligned}
\qquad
P_2 = \{S_2 \to bS_2\}
\qquad
P_3 = \{S_3 \to cS_3\}
$$

Tento systém generuje jazyk $L(\Gamma) = \{a^nb^nc^n \mid n \geq 1\}$ [5]. Je patrné, že tento systém vyhovuje definici 6.1, protože jeho master komponenta neobsahuje žádné pravidlo, ve kterém by bylo víc než jeden neterminál, a obě zbývající komponenty pouze genereují řetezec znaku b respektive c, v každé derivaci právě jeden symbol. Přitom i přes všechna tato omezení generuje tento systém jazyk, který není bezkontextový — generativní síla tohoto systému dalece přesahuje sílu jeho jednotlivých komponent.

Kapitola 7

Syntaktická analýza

Syntaktická analýza je proces, při kterém je analyzován řetězec symbolů s cílem zjistit, zda jeho struktura odpovídá určitému formálnímu jazyku. Takový jazyk můžeme definovat gramatikou, která generuje řetězce z daného jazyka, nebo automatem, který naopak přijímá řetězce tohoto jazyka. Je patrné, že pro proces syntaktické analýzy nás budou zajímat především automaty, které představují potřebný výpočetní model.

Jako základ pro syntaktickou analýzu tedy vužijeme PC systémy zásobníkových automatů. Tyto systémy jsme popsali v 4 a dále jsme se také v 4.1 zaměřili na problémy, které způsobuje nedeterminismus v těchto systémech. Nedeterminismus je syntaktickou analýzou obecně sepjatý problém, který ji může výrazně zkomplikovat, respektive prodražit, či dokonce efektivně znemožnit.

V rámci této práce budeme uvažovat i nedeterministické systémy, které přes své nevýhody nabízejí prostředek pro analýzu některých potencionálně zajímavých jazyků. Protože model PC systémů, ať již gramatických či automatových, zavádí poměrně mohutnou vyjadřovací schopnost, vyvstává otázka, zda tyto systémy nejsou až příliš silné. Tohoto tématu se například pro gramatické systémy dotýká článek [1]. Budeme tedy pracovat s premisou, že hlavní potenciál využití PC systémů v praxi leží v doplnění stávajích používaných modelů, které je většinou založené na bezkontextových jazycích, o silnější model, který je schopen řešit některé zvláštní případy, a ne v jejich úplném nahrazení. Můžeme o nich tedy smýšlet jako možných komponentách jiných, větších, metasystémů.

Například do programovacích jazyků, které bývají z drtivé většiny bezkontextové, jsou totiž často z různých důvodů zaváděny konstrukce, které nejsou bezkontextové. Často tyto struktury slouží pro zjednodušení práce programátora. Takovou konstrukcí může být například příkaz hromadného přiřazení (`x, y = 1, 2`), který zhruba odpovídá příkladu 4.1, pokud budeme uvažovat, že a a b představují identifikátory či literály, a c odpovídá rovnítku. Tyto konstrukce se dnes analyzují za využití několika úrovní překladače či kompilátoru, protože samotný syntaktický analyzátor tuto konstrukci nedokáže úplně zpracovat.

Zpočátku si ale představíme náš návrh algoritmu pro převod omezených lineárních PC gramatických systémů, které jsme si zavedli v 6.1, na PC systém zásobníkových automatů.

7.1 Algoritmus převodu

Následující algoritmus umožňuje převod omezeného lineárního PC gramatického systému na PC systém zásobníkových automatů. Tento převod je rozdělen na dvě hlavní fáze — převod master komponent, a následně převod všech ostatních komponent. Master komponenty

jsou převáděny pomocí algoritmu, který je znám pro převod bezkontextových gramatik a zásobníkové automaty. Všechny ostatní komponentní gramatiky jsou poté převedeny tak, že jejich protějšky v podobě zásobníkových automatů se chovají jako konečné automaty ukládající vybrané symboly na zásobník. U výsledného systému také platí to, že využívá modifikace, která zavádí implicitní přechody komponent (5.1).

Algoritmus 1 Převod omezeného lineárního PC gramatického systému na PC systém zásobníkových automatů

Vstup: Omezený lineární PC gramatický systém $\Gamma = (N, K, T, G_1, \ldots, G_n)$
Výstup: PC systém zásobníkových automatů $\mathcal{A} = (\Sigma, \Gamma, K, A_1, \ldots, A_n)$

1: $\Sigma = T$
2: $\Gamma = N \cup K \cup T$

3: $A_1 = (\{s_1\}, \Sigma, \Gamma, \delta_1, s_1, Z_1, \emptyset)$ ▷ master komponenta

4: **for each:** $a \in \Sigma$ **do**
5: $\quad \delta_1(s_1, a, a) = \{(s_1, \varepsilon)\}$
6: **end for**

7: **for each:** $A \to x \in P_1$ **do**
8: $\quad \delta_1(s_1, \varepsilon, A) = \{(s_1, x)\}$
9: **end for**

10: **for** $i = 2, n$ **do** ▷ zbývající komponenty
11: $\quad A_i = (\{s_i\}, \Sigma, \Gamma, \delta_i, s_i, Z_i, \{s_i\})$

12: $\quad \delta_i(s_i, x, Z_i) = \{(s_i, QxZ_i)\}$ ▷ $A \to xQB \in P_i$ (6.1)
13: $\quad \delta_i(s_i, x, x) = \{(s_i, Qxx)\}$

14: $\quad$ **for each:** $a \in \Sigma, a \neq x$ **do**
15: $\quad\quad \delta_i(s_i, a, Z_i) = \{(s_i, Z_i)\}$
16: $\quad\quad \delta_i(s_i, a, x) = \{(s_i, x)\}$
17: $\quad$ **end for**

18: $\quad$ **if** x se vyskytuje na pravé straně pravidla jiné komponenty **then**
19: $\quad\quad \delta_i(s_i, x, Z_i) = \{(s_i, Z_i)\}$
20: $\quad\quad \delta_i(s_i, x, x) = \{s_i, x\}$
21: $\quad$ **end if**
22: **end for**

Popišme si nyní tento algoritmus více prodrobněji. Začněme jeho druhou částí. Každý výsledný zásobníkový automat funguje jako konečný automat akceptující řetězec generovaný gramatikou, podle které byl vytvořen. Symboly tohoto řetězce si ukládá na zásobník tak, aby si jej mohly vyžádat ostatní komponenty. Zde je důležité upozornit, že pokud se v systému nachází alespoň jedna další komponenta, která je schopna takový symbol generovat, je nutné navíc zavést přechody, které také umožní přečtení tohoto symbolu bez jeho vložení na zásobník. Důvodem je, že zatímco všechny ostatní symboly může automat pouze přečíst (slouží pouze pro jeho synchronizaci), u takovýchto symbolů není jasné, zda

jsou synchronizační či ne. Chybné uložení synchronizačního symbolu na zásobník může vést k neschopnosti master komponenty akceptovat vstupní řetězec. Tato potřeba je největší slabinou tohoto algoritmu, neboť do výsledného systému zavádí nedeterminismus. Pokud se nachází v některém pravidle gramatiky dotazovací symbol, jemu odpovídající přechody jej zavedou na vrchol zásobníku.

Vraťme se k první části algoritmu, která provádí převod master komponenty. Vidíme zde zavádení dvou druhů přechodů. Prvním je, podle pravidel master gramatiky, expandován vrchol zásobníku. Pomocí druhého je takto získaný obsah zásobníku porovnáván se vstupním řetězcem. Tomuto principu se říká syntaktická analýza shora dolů — na zásobníku je postupně z počátečního symbolu generován řetězec jazyka, který je průběžně porovnáván se vstupem.

Výsledný systém také zachovává vlastnosti toho zdrojového. Pokud byl zdrojový systém centralizovaný, platí to samé i pro výsledný. Totéž platí i pro navracející se systémy. Pozorný čtenář si také jistě povšimnul, že výsledný systém kombinuje módy, ve kterých jeho komponenty akceptují vstupní řetězec. V případě master komponenty jde o vyprázdění zásobníku, u ostatních je to dosažením koncového stavu. Jelikož je jediný stav ne-master komponent zároveň jejich koncovým, rozhodnutí o přijetí či nepřijetí vstupního řetězce je zcela v režii master komponenty.

7.2 Koncept analýzy

Již jsme si řekli, že PC systémy zásobníkových automatů představují vhodný výpočetní model pro použití při syntaktické analýze. Ideálním je takový systém, který je deterministický. Takový systém může z libovolné konfigurace provést žádný, nebo právě jeden přechod. Porušení této podmínky lze poté považovat za chybový stav, který ukončí proces syntaktické analýzy s neúspěchem. K těmto systémům se ale váže komplikace spojená s komunikací mezi jednotlivými komponentami — každá komponenta, která někdy požaduje komunikaci, musí také v definici svých přechodů zohlednit to, že se na jejím zásobníku mohou vyskytnout i symboly pocházející z jedné či více jiných komponent. V případě PC gramatických systémů bychom navíc museli zavést ještě silnější omezení než ta představená v 6.1.

Další možností jak se s nedeterministickými systémy vypořádat, je analýza všech větvení, která takový systém vygeneruje. To je známo také jako prohledávání stavového prostoru [11]. Jednotlivé konfigurace systému představují *uzly grafu stavového prostoru.* Přechody mezi konfiguracemi, uzly, jsou *hrany* grafu — každá odpovídající právě jednomu přechodu. Počáteční konfigurace systémy představuje *počáteční uzel* grafu, konfigurace, v nichž dojde k odmítnutí či přijetí vstupního řetězce jsou *koncové uzly.* Jako *cestu* grafem budeme považovat takovou posloupnost uzlů a hran, která vede od počátečního uzlu ke koncovému. Pokud v tomto uzlu došlo k přijetí vstupu, představuje taková cesta posloupnost přechodů, pomocí kterých došlo k přijetí vstupního řetězce. Pokud nedojde k nalezení žádného koncového uzlu, ve kterém dojde k přijetí vstupního řetězce, můžeme použít cesty k uzlům, kde dojde k jeho odmítnutí, k identifikaci syntaktické chyby.

Existuje mnoho algoritmů pro prohledávání stavového prostoru. Pro naši potřebu bude ale vhodná taková metoda, která je *úplná* [11] (pokud existuje nějaké řešení, metoda jej nalezne) a *optimální* [11] (nalezené řešení je to s nejkratší cestou mezi počátečním a koncovým uzlem). Tyto podmínky splňují dvě metody, jmenovitě *prohledávání do šířky* a *prohledávání se zpětným návratem (backtracking)* [11]. *Backtracking* nabízí zajímavý potenciál pro optimalizace, nicméně jeho implementace je komplikovanější — existuje zde riziko nechtěné implementace metody *prohledávání do hloubky* [11] místo backtrackingu. Prohledávání do

hloubky ovšem není ani úplná ani optimální metoda. Zvolíme si tedy metodu prohledávání do šířky.

7.3 Analýza

Základními prvky syntaktické analýzy bude množina zásobníkových automatů a řídící jednotka, zajišťující jejich synchronizaci a komunikaci. Pro potřeby metody prohledávání do šířky k nim navíc přidáme frontu, ve které budou uchovávány jednotlivé dosud nezanalyzované konfigurace, uzly. Zásobníkové automaty využijeme pro aplikace přechodové funkce na příslušnou konfiguraci — ta je součástí konfigurace celého systému, proto si námi použité zásobníkové automaty nepotřebují samy uchovávat aktuální stav. Tato vlastnost přináší i výhodu v situacích, kdy dojde k nedeterministickému přechodu — výsledné konfigurace může uchovat řídící jednotka, bez potřeby vytvářet při každém větvení kopie takového automatu s jiným stavem.

7.3.1 Řídící jednotka

Tato jednotka tvoří jádro celého systému. Na základě analýzy stavu komponent systému rozhoduje, jaká další operace se provede. Zde je jejich výčet:

- akceptující krok
- komunikační krok
- přijmutí vstupního řetězce
- odmítnutí vstupního řetězce aktuální větví
- odmítnutí vstupního řetězce

Pokud dojde k nalezení takové konfigurace, ve které všechny komponenty systému přijaly vstupní řetězec, je také přijat systémem. Výstupem řídící jednotky v takovémto případě musí být informace o posloupnosti přechodů, jakými byla taková konfigurace dosažena. Pro tento účel si rozšíříme definici konfigurace PC systému zásobníkových automatů (4.2) o další prvek τ. Ten bude představovat konfiguraci, ze které byla ta současná dosažena právě jedním krokem. Tím vznikne zpětně vázaný lineární seznam, z něhož dokážeme rekonstruovat posloupnost konfigurací, jakých systém nabýval. Dále si tuto definici rozšíříme tak, aby konfiguraci jedné komponenty představovaly čtyři prvky. První tři ponecháme beze změny a přidáme další, ρ, který obsahuje hodnotu přechodové funkce dané komponenty, jakou byla taková konfigurace dosažena. Pokud tato hodnota neexistuje, tedy například proto, že této konfigurace bylo dosaženo komunikací, je ρ prázdné. Výsledná konfigurace $PCPA(n)$ bude tedy vypadat následovně

$$(s_1, x_1, \alpha_1, \rho_1, \ldots, s_n, x_n, \alpha_n, \rho_n, \tau).$$

Pomocí těchto dodatečných hodnot jsme schopni plně zrekonstruovat postup systému, jakým byl vstupní řetězec přijat.

Akceptující krok je základní krok systému, jedná se o jeho přechod do nové konfigurace. Jak jsme si již v 4.1 popsali, takovýchto konfigurací může být i vícero. Proto zde využijeme metody prohledávání do šířky. Všechny konfigurace, do kterých může systém přejít, uložíme postupně do pomocné fronty. V každém kroku vždy vyjmeme první položku

z fronty, která bude nyní aktuální konfigurací systému, a provedeme patřičnou operaci. Pokud dojde k vyprázdění fronty a nenalezení konfigurace, která by přijala vstup, dochází k odmítnutí vstupního řetězce.

Je-li v aktuální konfiguraci systému alespoň u jeho jedné komponenty přítomen dotazovací symbol na vrcholu jejího zásobníku, dochází k iniciaci komunikačního kroku. Ten má vždy přednost před akceptujícím. Před samotnou komunikací je nutné provést kontrolu její uskutečnitelnosti. Komunikace je neuskutečnitelná, dojde-li ke kruhovému dotazu. V takovém případě dojde buď k zacyklení systému, protože se bude neustále pokoušet provést nemožný komunikační krok, a nebo je možné ukončit proces s chybou. Kruhový dotaz lze identifikovat následně:

1. vytvoř množinu *žadatelé* všech komponent, které iniciují komunikaci
2. vytvoř množinu *žádaní* všech komponent, na které je směřován dotaz
3. *proveditelné* = *žadatelé* − *žádaní*
4. pokud je *proveditelné* = $\emptyset$, došlo ke kruhovému dotazu

Zároveň množina *proveditelné* představuje podmnožinu všech dotazů, kterou lze realizovat v tomto komunikačním kroku. Také je důležité nezapomenout, že v případě navracejících se systémů dochází k redukci zásobníku dotazované komponenty až po uspokojení všech dotazů v daném kroku. Komunikační krok se opakuje tak dlouho, než se splní všechny požadavky na komunikaci a nebo nedojde k zablokování systému.

V případě, že narazíme na takovou konfiguraci, ze které nelze provést žádný krok do jiné, uložíme si tuto konfiguraci do pomocného seznamu neúspěšných větví. Dojde k odmítnutí vstupu aktuální větví, a protože nelze provést nijaký krok, tato větev zaniká. V případě, že dojde k systémovému odmítnutí vstupu, vyjmeme z tohoto seznamu takovou konfiguraci, ve které byl vstup přečten z největší části. Tedy tu, kde $\sum_{i=1}^{n} |x_i|$ je nejmenší. Pokud je takových konfigurací několik, vybereme tu první. Protože v této konfiguraci došlo k přečtení největší části vstupního řetězce, budeme předpokládat, že ostatní větve byly neúspěšné z jiných důvodů, a pouze v této vybrané větvi byla nalezena syntaktická chyba. Tato konfigurace je také výstupem syntaktické analýzy, neboť informace v ní obsažené je možné použít pro opravu chyby.

7.3.2 Komponenty

U komponent PC systémů zásobníkových automatů budeme v námi navrhované metodě využívat modifikaci zavedenou v 5.1. Důvody pro její zavedení jsme popsali v 4.1. To klade na komponenty, se kterými budeme pracovat, několik požadavků. Prvním je schopnost identifikovat, zda-li je možné uplatnit implicitní přechod. Tím je implikován i druhý požadavek — uplatnění takového přechodu. Je tedy nutné, aby byla komponenta schopna dynamicky za běhu upravit svoji přechodovou funkci.

7.3.3 Paralelizace

Protože pracujeme s *paralelně* komunikujícími systémy, nabízí se otázka využití paralelizace i v syntaktické analýze založené na těchto systémech. Námi navržená metoda nabízí hned dvě možnosti, kde ji lze uplatnit. Zaprvé se jedná o paralelizaci komponentních výpočtů. Výpočty potřebné pro výpočet nových konfigurací, jakých mohou komponenty dosáhnout,

lze provádět paralelně, za předpokladu, že tyto procesy jsou synchronizovány v každém kroku. Tato potřeba synchronizace značně omezuje přínos paralelizace, obzvláště u menších systémů. Zde se skrývá zajímavý problém pro budoucí výzkum — možnost predikce komunikačních kroků. Potřeba synchronizovat komponenty po každém kroce totiž plyne právě z potencionální potřeby provést komunikační krok. Pokud by bylo možné určit, za jaký počet kroků dojde ke komunikačnímu kroku, bylo by možné nechat každou komponentu provést právě tolik kroků před tím, než by řídící jednotka zahájila synchronizaci celého systému. V takovémto případě by paralelizace komponent nabyla většího významu.

Druhou oblastí, kde je možné využít paralelizaci, je metoda prohledávání do šířky. Namísto použití fronty pro jednotlivé větve bychom mohli ihned spouštět procesy, které by tyto větve analyzovaly. Zde by bylo předmětem synchronizace předávání výsledků z jednotlivých stavových podprostorů těm nadřazeným, až po hlavní. Přínos paralelizace v tomto případě bude závislý na konkrétním systému. U systémů, které neobsahují větší množství nedeterminismu, nebo těch, jejichž slepé větve rychle zaniknou, může režie způsobená zavedením paralelismu způsobit, že se celý proces naopak zpomalí. Protože zde uvažujeme takový syntaktický analyzátor, který není specificky navržen pro analýzu konkrétního jazyka, není možné objektivně rozhodnout, zda by využití paralelizace mohlo urychlit jeho činnost. Z tohoto důvodu se přikláníme k jeho sekvenční variantě, neboť umožňuje přímočařejší implementaci. Použití paralelizace doporučejeme uvážit v případě analyzátoru pro nějaký konkrétní jazyk.

Kapitola 8

Implementace

Podle návrhu metody syntaktické analýzy z předchozí kapitoly byl implementován syntaktický analyzátor v podobě konzolové aplikace, v objektově orientovaném jazyce C# pro framework Mono/.NET ve verzi 4.5. Tento jazyk jsme zvolili zejména pro jeho možnosti práce s nejrůznějšími kolekcemi, hlavně pomocí provádění dotazů nad nimi použitím LINQ. Jeho objektová orientovanost také umožnila přenést teoretický syntaktický analyzátor z předchozí kapitoly do praxe bez potřeb změn jeho struktury. Celá aplikace je koncipována jako *simulátor* PC systémů zásobníkových automatů. Aplikace umožňuje načíst uživatelem vytvořený popis takového systému ve formátu XML a následně může simulovat jeho činnost při přijímání vstupu zadaného uživatelem. Výstupem je sekvence konfigurací, kterými procházela každá komponenta systému, a to včetně hodnot přechodové funkce, kterými byly dosaženy, je-li taková informace dostupná.

8.1 Struktura aplikace

Celá návrh aplikace je rozvržen v několika třídách, z nichž některé zajímavější si zde uvedeme včetně popisu jejich činnosti.

- `PushdownAutomaton` reprezentuje zásobníkový automat. Uchovává hodnoty jako je přechodová funkce, počáteční stav a počáteční zásobníkový symbol. Obsahuje logiku pro provedení přechodu z jedné konfigurace do druhé a také pro komunikaci se zásobníky. Důležitou částí je také implementace implicitních přechodů.

- `PCPASystem` představuje řídící jednotku. Zajišťuje provádění potřebných kroků systému, synchronizaci komponent a zprostředkovává jejich komunikaci. Objekty této třídy se vytvářejí na základě specifikace systému, kterou poskytne uživatel. Zajišťuje také uchovávání stavu systému a implementuje metodu prohledávání do šířky, která je použita pro simulaci nedeterministických systémů.

- `PCPAConfiguration` představuje konfiguraci systému. Jednotlivé konfigurace mezi sebou vytvářejí zpětně vázaný lineární seznam, pomocí kterého lze rekonstruovat postup analýzy. Výstupem syntaktické analýzy je právě taková konfigurace, ve které systém řetězec akceptoval a nebo ve které předpokládá synaktickou chybu.

8.2 Specifikace systému

Systém, který chce uživatel simulovat, je zadán XML souborem. Tento formát byl zvolen proto, že se jedná o známý a zavedený systém pro ukládání dat. To umožňuje především jeho validaci i zpracování bez potřeby vytvářet další analyzátor. S aplikací je distribuováno i několik ukázkových souborů. Část z nich je vytvořená podle příkladů uvedených v této práci. Příklad, jak takovýto soubor může vypadat, je také uveden v příloze A.

Kapitola 9

Závěr

Gramatiky i automaty patří neodmyslitelně do teorie formálních jazyků. Oba tyto modely nám umožňují popisovat svět okolo nás způsobem, jakému jsou schopny porozumět stroje. Ty jsou našimi každodenními společníky již dnes, a tento stav se den za dnem, objev za objevem prohlubuje. Stroje jsou neustále zdokonalovány, s impozantním cílem dosáhnout strojového ideálu — umělé inteligence. Takový ideál je pochopitelně ovlivněn těmi, kdo se o jeho dosažení pokoušejí — námi, lidmi. S člověkem je neodmyslitelně spojena komunikace. Během naší historie se objevilo nesčetně forem komunikace, mezi něž samozřejmě patří i jazyky. Mluveným i psaným slovem jsme schopni formulovat naše myšlenky, od těch nejmalichernějších až po ty nejabstraktnější teorie. Co je ale ještě zásadnější, je to, že jsme tyto myšlenky takto schopni předávat i jiným lidem a naopak od nich přijímat ty jejich. Je tedy jen pochopitelné, že bychom chtěli být schopni podobně, ne-li právě stejně, komunikovat i se stroji. O to právě usiluje teorie formálních jazyků. Neustále je posouván výzkum dále a s každým objevem, který nám dovoluje popsat nebo analyzovat čím dál složitější systémy, jazyky, se přibližujeme ke schopnostem stroje, který je prozatím schopen těch nejsložitějších operací s jazyky — našemu mozku. A kdo ví, jednou nás možná naše výtvory i předčí.

I pro systémy, ať už gramatik či automatů, které jsme si na počátku této práce představili, není těžké vypozorovat jejich inspiraci lidským světem. Nestačí-li na vyřešení nějakého problému člověk sám, je pravděpodobné, že takovýto problém může být vyřešen týmem spolupracujících lidí. Každý takový člověk je obdařen jinými schopnostmi, má jiné znalosti i náhled na řešený problém. A je to právě jejich spolupráce, která vede k zapojení jejich schopností tak, že takový tým je jako celek schopen dosáhnout výsledků, jakých by žádný jeho člen sám nedosáhnul. Toto vystihuje následující citát, který je připisován Aristotelovi.

> *„Celek je větší než suma jeho částí.“*

Identická myšlenka provází i gramatické a automatové systémy, pouze na ně nahlížíme více z matematického než-li filozofického hlediska. Takovéto zvyšení síly těchto systémů jsme si názorně ukázali i na několika příkladech sytémů, které popisovaly jazyky, které nebyly bezkontextové, a přitom jejich samotné komponenty byly nanejvýše bezkontextové, ne-li slabější.

V této práci jsme se zabývali paralelně komunikujícími systémy. Tyto systémy jsou tvořeny částmi, říkáme jim komponenty, které pracují paralelně a synchronně. Každá komponenta buď vytváří vlastní větu, to v případě systémů gramatických, nebo se naopak, u automatových systémů, pokouší přijmout vlastní kopii vstupní věty. V obou systémech je také přítomna jedna určitá komponenta, unikátní ne svými schopnostmi, ale svým postavením, říkáme jí *master komponenta*. V případě gramatických systémů můžeme o této

součástce pronést, že zastává exekutivní pozici, protože je to právě takový jazyk, jaký tato komponenta, za přispění ostatních, generuje, jenž je jazykem celého systému. U automatových systémů toto zcela neplatí, neboť tyto systémy akceptují vstup právě tehdy, je-li přijat všemi jeho komponentami. Ovšem je samozřejmě možné vytvořit takový automatový systém, kde rozhodnutí o přijmutí vstupu leží efektivně zcela na master komponentě. Takový systém je vytvořen algoritmem pro převod omezených lineárních paralelně komunikujících gramatických systémů, jenž jsme představili v kapitole 6, právě na paralelně komunikující systémy zásobníkových automatů.

Master komponenty hrají důležitou roli také v určitých variantách gramatických i automatových systémů. Máme samozřejmě na mysli takzvané *centralizované* systémy. V takovýchto systémech platí, že právě master komponenta takového systému může jako jediná iniciovat komunikaci s jinými komponentami. Centralizované nejsou jedinou variantou těchto systémů, kterou jsme v této práci zmínili. Uvedli jsme si také takové systémy, o kterých říkáme, že jsou *navracející se*. V těchto systémech je definováno speciální chování jejich komponent po proběhnutí komunikace. Komponenta, na kterou byl směřován požadavek na komunikaci, po jeho uspokojení zavrhne výsledek své dosavadní práce a začne znovu od samého začátku. U gramatických systémů je to rovno nahrazení její stávající vytvořené věty startovacím neterminálním symbolem. Pro komponenty paralelně komunikujících systémů zásobníkových automatů, jakožto jediného představitele automatových systémů, kterého jsme uvažovali, dojde k vyprázdění celého jejich zásobníku a následného uložení příslušného počátečního symbolu na něj.

Protože cílem této práce bylo vytvořit syntaktický analyzátor schopný zpracovat i jazyky, které nejsou bezkontextové, zaměřili jsme se především na paralelně komunikující systémy zásobníkových automatů, které představují výpočetní model potřebný právě pro syntaktickou analýzu. V souvislosti s tím jsme si také popsali problémy, jaké se pojí s těmito systémy pokud jde o nedeterminismus — jejich nejednoznačnost. Ukázali jsme si, že jeho dopad je bohužel zesílen stejným způsobem jako jejich síla. Konkrétně k tomutu zesílení dochází v situacích, kdy může provést vícero komponent současně nedeterministický krok. Tento problém jsme dále sledovali a popsali jsme si jednu konkrétní situaci, kdy je do součástí systému zaváděn nedeterminismus ne jako prostředek pro popis nějakého jazyka, ale jako řešení problému spojeného s činností komponent v systému. Tímto problémem je, že obecně může být po každé komponentě vyžadováno, aby byla schopna pokračovat v práci i poté, co byl již naplněn její logický účel, právě proto, že některé komponenty svoji práci ještě nedokončily. Jako řešení tohoto problému jsme zavedli nový druh přechodu pro zásobníkové automaty, které jsou součástí paralelně komunikujících systémů, takzvaný *implicitní přechod*. Věříme, že tento druh přechodu může být použit jako prakticky aplikovatelné řešení umožňující nejen zjednodušit návrh zmíněných systémů, ale také zejména omezit množství v nich obsažené nejednoznačnosti, a přeneseně tedy i zvýšit jejich efektivnost.

V předchozím textu jsme se také již zmínili o algoritmu převádějícím gramatické systémy na jejich automatové protějšky. Ještě před tím jsme definovali jistá omezení pro gramatické systémy. Systémy splňující tato omezení jsme nazvali omezené lineární paralelně komunikující gramatické systémy. Tyto systémy jsou tvořeny komponentami, jenž jsou tvořeny podmnožinou regulárních gramatik a jejichž master komponenta je gramatika lineární. Příkladem jsme dokázali, že i takto silně omezené gramatické systémy jsou schopné generování jazyků, které nejsou bezkontextové. A právě pro takto regulované systémy jsme předložili návrh algoritmu, pro jejich převod na paralelně komunikující systém zásobníkových automatů.

Na samém konci práce jsme navrhli a diskutovali metodu použití paralelně komunikujích

zásobníkových automatů pro syntaktickou analýzu. Uvedli jsme, že problém nedeterminismu lze řešit jeho převedením na problém prohledávání stavového prostoru. Pro jeho řešení je možné využít několika metod, z nichž jsme zmínili celkem tři konkrétní. U jednotlivých metod jsme zvážili jejich vhodnost a nakonec jsme zvolili metodu, která se jevila jako nejlepší. Tím jsme dosáhli toho, že náš syntaktický analyzátor je silnější, než kdybychom jej omezili pouze na deterministickou analýzu. S tímto hotovým rozhodnutím jsme se přesunuli k samotným paralelně komunikujícím systémům zásobníkových automatů.

Ty jsme si rozdělili na dvě části — řídící jednotku a samotné komponenty. Řídící jednotka tvoří mozek celého systému, řídí jeho komponenty a zprostředkovává jejich komunikaci. Také jsme kvůli syntaktické analýze upravili definici konfigurace systému tak, aby nám umožnila uchovat informace o tom, jakými konfiguracemi systém postupně procházel a za pomoci jakých přechodů, jednalo-li se o nějaké, toho dosáhl. Abychom si ulehčili práci s nedeterministickými systémy, navrhli jsme použití bezstavových komponent — konfigurace jednotlivých automatů uchovává řídící jednotka a jednotlivé automaty tvoří pouze logiku pro výpočet konfigurací, do kterých může komponenta přejít. Tím dosáhneme toho, že v nedetermistických systémech nám postačí vytvářet kopie konfigurací a ne celých automatů. Také jsme postupně popsali všechny operace, jaké musí být řídící jednotka schopna provést, a co tyto akce obnášejí. A nakonec jsme si popsali, jakým způsobem identifikovat syntaktickou chybu v nedeterministickém systému.

Protože většina logiky systému je pokryta řídící jednotkou, u jejich komponent jsme uvedli pouze požadavek na jejich schopnost dynamicky měnit svoji přechodovou funkci, protože ve výsledném analyzátoru plánujeme využít implicitních přechodů. Námi navrhnutou metodu syntaktické analýzy jsme zakončili diskuzí o možné paralelizaci výpočtů v systému. Kloníme se spíše k názoru, že rozhodnutí o paralelizaci by mělo být učiněno na základě konkrétního analyzátoru pro konkrétní jazyk. Navíc místo paralelizace jednotlivých komponent se jeví jako přínosnější paralelizace metody prohledávání stavového prostoru. Zde také vyvstal zajímavý směr pro budoucí výzkum, který si včetně několika dalších uvedeme.

V této práci jsme uvažovali pouze synchronizované systémy. Tyto systémy jsou zajímavé svojí silou, která je primárně důsledkem jejich synchronizace. U těchto systémů je nutné, aby každá komponenta provedla vždy právě jeden krok a poté je nutné, aby řídící provedla kontrolu, zda nevznikl požadavek na komunikaci. Další krok, derivační u gramatických a akceptující u automatových systémů, lze provést pouze po uspokojení všech požadavků na komunikaci. Pokud by byla řídící jednotka schopna predikovat, za jaký počet kroků by došlo k dalšímu požadavku na komunikaci, bylo by možné úplně zanedbat synchronizaci mezi dvěma komunikačními kroky — pouze by stačilo, aby řídící jednotka, po ukončení posledního ze sekvence po sobě jdoucích komunikačních kroků, provedla predikci další komunikace a její výsledek předala všem komponentám. Každá komponenta by tak mohla provést příslušný počet generujících či akceptujících kroků plně samostatně a bez rizika ztráty synchronizace se zbytkem systému. Řídící jednotka by vyčkala na dokončení práce jednotlivých komponent a poté uskutečnila další komunikační krok a celý cyklus by se opakoval. Takto by bylo dosaženo snížení zátěže způsobené nepotřebnou synchronizací a tím by i došlo ke zlukrativnění paralelizace samotných komponent. Schopnost předpovídání komunikace by také otevřela další možnosti — například identifikace komponent, na které již nevznikne žádný dotaz. Takovéto komponenty by mohly předčasně ukončit svoji činnost.

Dalším zajímavým problémem je formalizování definice výskytu syntaktické chyby v nedeterministických systémech zásobníkových automatů. Pokud zanedbáme možnost chyby v návrhu, pak u deterministického systému je neschopnost provést krok indikátorem syntaktické chyby. U jejich nedeterministických variant to ale může znamenat pouze to, že

aktuálně prohledávaná větev stavového prostoru je jenom slepá. Určení syntaktické chyby v konfiguraci, ve které byla přečtena nejdelší část vstupního řetězce, je sice intuitivní, ale nezaručuje, že vždy dojde k této identifikaci správně. Představme si situaci, kdy syntaktická chyba ve vstupním řetězci způsobí, že díky ní dokáže některá prohledávaná větev mylně zpracovat delší část vstupu než ta větev, která takovou chybu správně odhalila. V takovéto situaci dojde k nesprávné identifikaci chyby, čímž dojde ke znehodnecení jakéhokoliv výstupu syntaktického analyzátoru.

Na úplný závěr této práce si uveďme poslední zbývající otevřený problém. Tím je absence důkazu o ekvivalenci vstupního a výstupního systému námi navrženého algoritmu. To lze navíc i zobecnit na problém možnosti algoritmického převodu gramatických systémů na ekvivalentní automatové systémy. V [8] již byla zkoumána analýza paralelně komunikujících gramatických systémů pomocí jiného automatového modelu.

Literatura

[1] Bruda, S.; Wilkin, M.: Not Seeing the Parse Trees from the Parse Forest of a Context-Free Parallel Communicating Grammar System. In *Parallel and Distributed Computing (ISPDC), 2013 IEEE 12th International Symposium on*, June 2013, s. 169–176.

[2] Chomsky, N.: Three models for the description of language. *IRE Transactions on Information Theory*, 1956: s. 113–124, ISSN 0096-1000.

[3] Csuhaj-Varjú, E.; Martín-Vide, C.; Mitrana, V.; aj.: Parallel communicating pushdown automata systems. *International Journal of Foundations of Computer Science*, 2000: s. 631–650.
URL http://www.worldscientific.com/doi/abs/10.1142/S0129054100000338

[4] Csuhaj-Varjú, E.; Vaszil, G.: On context-free parallel communicating grammar systems: synchronization, communication, and normal forms. *Theoretical Computer Science*, 2001: s. 511–538, ISSN 0304-3975.
URL http://www.sciencedirect.com/science/article/pii/S0304397599003187

[5] Dumitrescu, S.; Păun, G.; Salomaa, A.: Pattern Languages Versus Parallel Communicating Grammar Systems. *International Journal of Foundations of Computer Science*, 1997: s. 67–80.
URL http://www.worldscientific.com/doi/abs/10.1142/S0129054197000057

[6] Martín-Vide, C.; Mateescu, A.; Mitrana, V.: Parallel finite automata systems communicating by states. *International Journal of Foundations of Computer Science*, 2002: s. 733–749.
URL http://www.worldscientific.com/doi/abs/10.1142/S0129054102001424

[7] Meduna, A.; Zemek, P.: *Regulated Grammars and Automata*. Springer US, 2014, ISBN 978-1-4939-0368-9, 694 s.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=10498

[8] Pardubská, D.; Plátek, M.: Parallel communicating grammar systems and analysis by reduction by restarting automata. *Linguistic and Formal Languages. Tarragona: Rovira Virgili University*, 2008: s. 81–98.
URL http://www.dcs.fmph.uniba.sk/~pardubska/publikacie/PCGS-RA-ForLing08-final.pdf

[9] Păun, G.: On the power of synchronization in parallel communicating grammar systems. *Stud. Cerc. Mat.*, 1989: s. 191–197.

[10] Păun, G.; Sântean, L.: Parallel communicating grammar systems: the regular case. *An. Univ. Bucureşti, Ser. Matem.-Inform.*, 1989: s. 55–63.

[11] Russell, S. J.; Norvig, P.: *Artificial intelligence: a modern approach (3rd edition).* Prentice Hall, 2009.

Příloha A

Formát souboru s definicí PC systému zásobníkových automatů

```
<?xml version="1.0" encoding="UTF-8"?>
<PushdownAutomataSystem IsReturning="false">
        <InputSymbols>
                <Symbol>a</Symbol>
        </InputSymbols>
        <PushdownSymbols>
                <Symbol>Z1</Symbol>
        </PushdownSymbols>
        <QuerySymbols>
                <Symbol Automaton="1">K1</Symbol>
        </QuerySymbols>
        <PushdownAutomaton AcceptingMode="FinalState" InitialState="s1"
            InitialStackSymbol="Z1">
                <States>
                        <State>s1</State>
                </States>
                <FinalStates>
                        <State>sf</State>
                </FinalStates>
                <Transitions>
                        <!-- přechod δ1(s1, a, Z1) = {(s1, aZ1)} -->
                        <Transition OldState="s1" InputSymbol="a"
                            TopmostSymbol="Z1" NewState="s1">
                                <Symbol>a</Symbol>
                                <Symbol>Z1</Symbol>
                        </Transition>
                        <!-- přechod δ1(s1, ε, Z1) = {(sf, ε)} -->
                        <Transition OldState="s1" TopmostSymbol="Z1"
                            NewState="sf" />
                </Transitions>
        </PushdownAutomaton>
</PushdownAutomataSystem>
```