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 STAVOVÝCH GRAMATIKÁCH

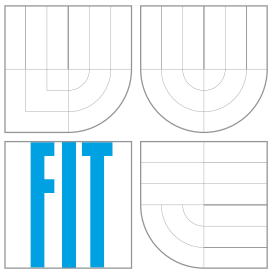
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MIROSLAV PAULÍK

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYNTAKTICKÁ ANALÝZA ZALOŽENÁ NA STAVOVÝCH GRAMATIKÁCH

PARSING BASED ON STATE GRAMMARS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MIROSLAV PAULÍK
AUTHOR

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

BRNO 2013

Abstrakt

Tato práce popisuje vlastnosti stavových gramatik a n–limitovaných stavových gramatik s důrazem na nedeterminismus v analýze takových gramatik. Zejména se zaměřuje na problémy způsobené povolením vymazávacích pravidel a možných výskytů rekurze. Na základě analýzy těchto problémů nabízí možná řešení, která jsou posléze uplatněna při návrhu prakticky zaměřené metody paralelní syntaktické analýzy. Tato metoda je výrazně rychlejší, než sekvenční analýza s návratem.

Abstract

This work describes the characteristics of state grammars and n–limited state grammars with a focus on non-determinism in a parsing proccess based on such grammars. In particular, it focuses on the problems caused by enabling erasing productions or by possible occur of recursion. This work also describes possible solutions to non-deterministic problems, which are used in the design of a parallel parsing method. This method is significantly faster than sequence analysis based on backtracking.

Klíčová slova

stavová gramatika, n–limitovaná stavová gramatika, determinizace, paralelní syntaktická analýza, hluboký zásobníkový automat

Keywords

state grammar, n–limited state grammar, determinisation, paralel parsing, deep pushdown automaton

Citace

Miroslav Paulík: Syntaktická analýza založená na stavových gramatikách, bakalářská práce, Brno, FIT VUT v Brně, 2013

Syntaktická analýza založená na stavových gramatikách

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Prof. RNDr. Alexandera Meduny, CSc.

.....
Miroslav Paulík
13. května 2013

Poděkování

Na tomto místě bych chtěl poděkovat mému vedoucímu, prof. Alexandru Medunovi, za odborné vedení, užitečné rady, které mi poskytl, ochotu a čas, který mi při tvorbě práce věnoval, a zejména za trpělivost, kterou se mnou měl.

Obsah

Kapitola 1

Úvod

Již v dobách prvních výpočetních strojů vyvstala otázka, jak těmto přístojům (počítačům) sdělit, co mají spočítat. Problém této komunikace tkví ve formální přesnosti a neměnnosti zadávaných pokynů směrem k počítači. Obecně ale platí, že běžnou řečí, přirozeným jazykem, se domluvit nedokážeme, neboť takový jazyk se v čase vyvíjí. Z tohoto důvodu vznikly tzv. „formální jazyky“, přesně popsané, ale do značné míry omezené jazyky. Je však snaha tyto formální jazyky vylepšovat tak, aby byly zároveň co nejjednodušší a měly co nejlepší vyjadřovací vlastnosti.

Jedním z výsledků této snahy bylo definování tzv. „stavových gramatik“ panem T. Kasaiem v roce 1970. Tyto gramatiky generují nekonečnou hierarchii jazyků, které se řadí svou vyjadřovací silou od bezkontextových až po kontextové jazyky [4]. Při zařazení těchto gramatik do Chomského hierarchie [6] však nebyla vzata v úvahu možná přítomnost tzv. „vymazávacích pravidel“, které ještě dále zvyšují generativní sílu těchto gramatik. V této práci se zaměřuji na popis metody syntaktické analýzy stavových gramatik, která právě tyto pravidla obsahuje. Mojí motivací pro tuto práci byla snaha vytvořit syntaktický analyzátor, který co nejrychleji a v konečném počtu kroků provede korektní analýzu. Cílem této práce je tedy navrhnout způsob analýzy tak, aby ji bylo možné v praxi co nejsnáze použít. V tomto duchu je vedena i struktura práce, kdy od teoretických podkladů v úvodu postupně přecházím přes popis vlastností stavových gramatik a návrh metodo analýzy až po poznatky z realizace prakticky zaměřeného analyzátoru.

Kapitola 2 této práce obsahuje základní pojmy a konstrukce z oblasti teorie formálních jazyků. Zejména jsou zde obsaženy formalismy jazyka, gramatik a automatů a patřičných matematických konstrukcí. Kapitola 3 je zaměřena na detailnější rozbor vlastností stavových gramatik. Zejména jsou zde popsány n–limitované stavové gramatiky a jejich zařazení v Chomského hierarchii jazyků. Kapitola 4 popisuje komplikace, jenž vyplývají z podstaty vlastností stavových gramatik a které je nutné zohlednit při návrhu metody syntaktické analýzy. Tímto návrhem se zabývá kapitola 5. V ní je nejprve nastíněno několik možných konceptů, z nichž je jeden vybrán. Ten je posléze doplněn tak, aby zohledňoval problémy popsané v kapitole 4. Vzniklá metoda je následně optimalizována tak, aby měla co nejlepší vlastnosti pro implementaci, která je popsána v kapitole 6. V závěru této práce je pak konečné shodnocení navržené metody a dále jsou zde diskutovány další možná rozšíření tohoto projektu. V přílohách této práce se pak nachází formát XML souboru gramatiky, popisky speciálních symbolů použitých v diagramech a kompletní příklad syntaktické analýzy jazykových konstrukcí, které již nejsou bezkontextové.

Kapitola 2

Prerekvizity

Tato kapitola popisuje pojmy a základní terminologii z oblasti teorie formálních jazyků, které jsem v této práci použil. V úvodu této kapitoly sekci 2.1 jsem popsal základní matematické konstrukce a operace. Sekce 2.2 obsahuje definice pojmů řetězec a jazyk, včetně patřičných operací s nimi spojených. Následující sekce, 2.3, obsahuje definice gramatik, tedy nástrojů pro generování jazyků, a jejich zařazení v rámci Chomského hierarchie. Závěrečná sekce 2.4 popisuje výpočetní modely pro tyto gramatiky, tzv. „automaty“. Celá tato kapitola čerpá zejména z [8, 6, 9].

2.1 Základní matematické konstrukce

2.1.1 Množiny

Množina M je soubor objektů, prvků množiny, které nemají žádnou strukturu vyjma členství. Abychom vyjádřili, že objekt x je prvkem množiny M, používáme zápis $x \in M$. Zápis $z \notin M$ naopak značí, že objekt x prvkem množiny M není. Definovat množinu lze mnoha způsoby. Nejsnáze ji lze popsat výčtem jejích prvků oddělených od sebe čárkou a uzavřených do společných složených závorek. Například množina M obsahující čísla 1, 2 a 3 se zapíše jako

$$M = \{1, 2, 3\}.$$

Je–li prvků množiny více a zároveň jsou následující prvky zřejmé, lze použít tzv. *výpůstek*. Malé písmena anglické abecedy lze tedy zapsat také jako $\{a, b, \ldots, z\}$. Je–li potřeba definovat prvky množiny více specifičtějším způsobem, např. tak, aby prvky množiny splňovaly vlastnost π, pak se používá následující notace

$$M = \{x | \pi(x)\}.$$

Pokud množina obsahuje konečný počet prvků, je označována jako *konečná množina*, v opačném případě se jedná o *množinu nekonečnou*. Počet prvků množiny, tzv. *kardinalitu* množiny, zapisujeme $|M|$. Chceme–li vyjádřit, že daná množina je prázdná, použijeme zápis $M = \emptyset$.

Mezi základní operace s množinami patří *sjednocení* ($\cup$), *průnik* ($\cap$) a *rozdíl* ($-$). Nechť M_1 a M_2 jsou množinami, pak

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

Pokud všechny prvky množiny M_1 jsou zároveň i prvky množiny M_2, pak M_1 je *podmnožinou* M_2, zapisováno jako

$$M_1 \subseteq M_2.$$

Pokud ale množina M_2 obsahuje alespoň jeden prvek x takový, že $x \notin M_1$, pak M_1 nazýváme *vlasní podmnožinou* M_2, zapisováno jako

$$M_1 \subset M_2.$$

V případě, že M_1 je podmnožinou M_2 a zároveň M_2 je podminožinou M_1, tedy platí $M_1 \subseteq M_2$ a $M_2 \subseteq M_1$, pak množiny M_1 a M_2 označujeme *identické*, zapisováno $M_1 = M_2$.

Množinu všech podmnožin množiny M označujeme 2^M, symbolicky zapisováno jako

$$2^M = \{P | P \subseteq M\}.$$

Multimnožina je rozšířením množiny tak, aby umožňovala vícenásobný výskyt stejného prvku (např. $\{1, 2, 2, 3\}$).

2.1.2 Relace a funkce

Rozšíříme–li definici multimnožiny o možnost definovat pořadí prkvů, dostaneme (uspořádanou) *sekvenci* prvků, nebo také *n–tici*. Sekvenci S obsahující prvky a a b zapisujeme jako

$$S = (a, b)$$

Nechť M_1 a M_2 jsou množiny, pak *kartézský součin* je definován jako

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

Pojmem *n–ární relace* je označován vztah mezi skupinou prvků jedné nebo více množin. *Binární relace* (dále jen relace) je označení pro vztah mezi právě dvěma množinami. Relace m z množiny M_1 do M_2 je podmnožinou kartézského součinu, tedy

$$m \subseteq M_1 \times M_2$$

Funkce ϕ z množiny M_1 do M_2, zapisováno jako $M_1 \to M_2$, je relace z M_1 do M_2 taková, že pro každé $x \in M_1$ platí

$$|\{y | y \in M_2 \text{ a } (x, y) \in \phi\}| \leq 1$$

2.2 Abeceda, řetězce a jazyky

Definice 2.1. *Abeceda* je konečná, neprázdná množina prvků, které nazýváme *symboly*.

Definice 2.2. Nechť Σ je abeceda, pak

1. ε[1] je *řetězec* nad abecedou Σ
2. pokud x je řetězec nad Σ a $a \in \Sigma$, potom xa je řetězec nad abecedou Σ

Definice 2.3. Nechť x je řetězec nad abecedou Σ, pak *délka řetězce* x, $|x|$, je definována

1. pokud $x = \varepsilon$, pak $|x| = 0$
2. pokud $x = x_1, \ldots, x_n$ pro $n \geq 1$ a $x_i \in \Sigma$ pro všechna $i = 1, \ldots n$, pak $|x| = n$

Definice 2.4. Nechť x a y jsou řetězce nad abecedou Σ, pak výsledkem *konkatenace* (zřetězení) x a y je nový řetězec xy.

Příklad 2.1. Konkatenace řetězců AAA a BBB je řetězec $AAABBB$.

Příklad 2.2. Konkatenace řetězců AAA a ε je řetězec $AAA\varepsilon = AAA$.

Definice 2.5. Nechť x je řetězec nad abecedou Σ. Pro $i \geq 0$, i–tá *mocnina řetězce* x, x^i, je definována

1. $x^0 = \varepsilon$
2. pokud $n \geq 1$, pak $x^n = xx^{n-1}$

Příklad 2.3. Nechť $x = AB$ je řetězec. Třetí mocninu řetězce x, x^3, získáme následovně

$$x^3 = xx^2 = xxx^1 = xxxx^0 = xxx\varepsilon = ABABAB$$

Definice 2.6. Nechť x a y jsou řetězce nad abecedou Σ. Pokud existuje řetězec z nad abecedou Σ takový, že platí $xz = y$, pak řetězec x je *prefixem* y.

Definice 2.7. Nechť x a y jsou řetězce nad abecedou Σ. Pokud existuje řetězec z nad abecedou Σ takový, že platí $zx = y$, pak řetězec x je *suffixem* y.

Definice 2.8. Nechť Σ^* značí množinu všech řetězců nad abecedou Σ. Každá podmnožina $L \subseteq \Sigma^*$ je *jazyk* nad abecedou Σ.

Σ^* je označení pro tzv. *univerzální jazyk*, tedy jazyk, který obsahuje všechny řetězce nad abecedou Σ. Obdobně, Σ^+ je označení pro všechny neprázdné řetězce nad abecedou Σ, přičemž platí

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

[1] Symbolem ε se v literatuře označuje prázdný řetězec.

2.3 Gramatiky a rodiny jazyků

Definice 2.9. *Neomezená gramatika*, nebo také typ 0, je čtveřice

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

kde

- N je abeceda neterminálů,
- $T \cap N = \emptyset$ je abeceda terminálů,
- $P \subseteq ((N \cup T)^* N (N \cup T)^*) \times (N \cup T)^*$ je množina pravidel gramatiky,
- $S \in N$ je počáteční neterminál.

Pravidla gramatiky, $r = (u, v) \in P$, jsou označována jako přepisovací pravidla (nebo také derivační pravidla). Místo zápisu $r = (u, v)$ používáme $r : u \to v$ nebo jen $u \to v$.

Rodina jazyků generovaná neomezenými gramatikami je rodinou všech *rekurzivně spočetných* jazyků **RE**.

Definice 2.10. *Kontextová gramatika*, označovaná jako typ 1, je neomezená gramatika

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

kde všechny pravidla $u \to v \in P$ mají následující tvar

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

kde $x_1, x_2 \in (N \cup T)^*$, $A \in N$ a $y \in (N \cup T^+)$.

Rodina jazyků generovaná kontextovými gramatikami je rodinou všech *kontextových jazyků*, které se v literatuře značí jako **CS**.

Definice 2.11. *Bezkontextová gramatika*, označovaná jako typ 2, je neomezená gramatika

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

kde všechny pravidla $r \in P$ jsou tvaru

$$A \to x,$$

kde $x \in (N \cup T)^*$ a $A \in N$.

Rodina jazyků generovaná bezkontextovými gramatikami je rodinou všech *bezkontextových jazyků*. Tato rodina jazyků je označována jako **CF**.

Definice 2.12. *Regulární gramatika*, označovaná jako typ 3, je neomezená gramatika

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

kde všechny pravidla $r \in P$ jsou tvaru

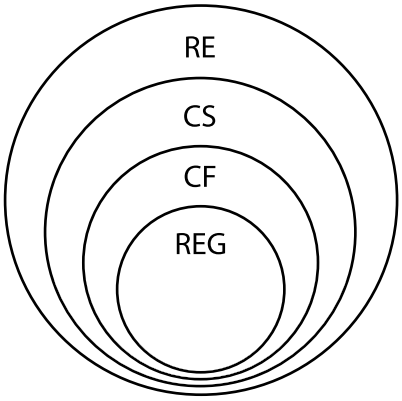
$$A \to aB \text{ nebo } A \to a,$$

kde $a \in T$ a $A, B \in N$.

Rodina jazyků generovaná regulárními gramatikami je rodinou všech *regulárních jazyků* a je označována jako **REG**.

Noam Chomsky, zakladatel teorie formálních jazyků, popsal a klasifikoval 4 základní kategorie jazyků a označil je jako typy 0–3 (viz definice 2.9 až 2.12) [12, 9, 6]. Jejich vzájemný vztah popisuje následující teorém a posléze demostruje i obrázek 2.3.

Teorém 2.13. **REG** $\subset$ **CF** $\subset$ **CS** $\subset$ **RE** (více informací viz [12, 9, 6]).



Obrázek 2.1: Chomského hierarchie jazyků [12].

2.4 Automaty

Tato sekce popisuje nástroje na rozpoznávání příslušnosti řetězců k patřičným (regulárním resp. bezkontextovým) jazykům. Jedná se o *konečný* a *zásobníkový automat*. Dále je zde popsán také *hluboký zásobníkový automat*, tedy modifikovaný zásobníkový automat, jenž umožňuje rozpoznávání řetězců i z některých dalších jazyků, které už nejsou bezkontextové.

Definice 2.14. *Konečný automat* je pětice

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

kde

- Q je konečná množina stavů
- Σ je vstupní abeceda
- $R \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ je množina pravidel
- $s \in Q$ je počáteční stav
- $F \subseteq Q$ je množina koncových stavů

Místo zápisu pravidla $(p, q, a) \in R$ používáme $pa \to q \in R$. *Konfigurace* automatu M je libovolný řetězec $\tau \in Q\Sigma^*$.

Definice 2.15. Nechť $pax, qx \in Q\Sigma^*$ jsou konfigurace a $pa \Rightarrow x \in R$, pak *derivační krok* z pax do qx je definován jako

$$pax \Rightarrow qx$$

Označení $\Rightarrow^k$ pro $k \geq 0$ vyjadřuje posloupnost k derivačních kroků. Není–li k shora omezené, pak zapisujeme $\Rightarrow^k *$. Platí–li navíc, že $k > 0$, pak používáme označení $\Rightarrow^+$.

Definice 2.16. *Jazyk přijímaný konečným automatem* M, $L(M)$, je definován jako

$$L(M) = \{w \in \Sigma^* | sw \Rightarrow^* f, f \in F\}.$$

Teorém 2.17. (viz [6]) *Jazyk L je regulárním právě a jen tehdy, když existuje konečný automat M takový, že $M = L(M)$.*

Konečný automat je výpočetní model, který ke své činnosti potřebuje pouze číst. Pokud bychom chtěli přidat funkcionalitu pro zápis, museli bychom přidat i paměťové úložiště, kam by mohl zapisovat. Takovým úložištěm může být třeba zásobník. Konečný automat se zásobníkem je definován následovně:

Definice 2.18. *Zásobníkový automat* je sedmice

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

kde

- Q, Σ, s, F jsou definovány stejně, jako u konečných automatů
- Γ je zásobníková abeceda obsahující speciální koncový symbol $\# \in \Gamma$
- $R \subseteq \Gamma \times Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^* \times Q$ je množina pravidel
- S je počáteční symbol na zásobníku

Analogicky s konečným automatem, místo zápisu pravidla $(A, p, a, w, q) \in R$ používáme $Apa \to wq \in R$. *Konfigurace* automatu M je libovolný řetězec $\tau \in \Gamma^* Q \Sigma^*$.

Definice 2.19. Nechť $xApay, xwqy \in \Gamma^* Q \Sigma^*$ jsou konfigurace a $Apa \Rightarrow wq \in R$, pak *derivační krok* z $xApay$ do $xwqy$ je definován jako

$$xApay \Rightarrow xwqy$$

Označení $\Rightarrow^k$ pro $k \geq 0$ vyjadřuje posloupnost k derivačních kroků. Není–li k shora omezené, pak zapisujeme $\Rightarrow^k *$. Platí–li navíc, že $k > 0$, pak používáme označení $\Rightarrow^+$.

Teorém 2.20. (viz [6]) *Jazyk L je bezkontextový právě a jen tehdy, když existuje zásobníkový automat M takový, že $M = L(M)$.*

Pro rozpoznávání jazyků, které obsahují konstrukce, které již nejsou bezkontextové, je potřeba ještě výkonější výpočetní model. Takový nástroj lze získat další modifikací zásobníkového automatu, např. úpravou přístupu do paměti. Příkladem modelu s takovou modifikací je *hluboký zásobníkový automat*.

Definice 2.21. *Hluboký zásobníkový automat* [7, 2] je sedmice

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

kde

- $Q, \Sigma, \Gamma, s, S, F$ jsou definovány stejně jako u zásobníkového automatu
- $R \subseteq (I \times Q \times (\Gamma - (\Sigma \cup \{\#\})) \times Q \times (\Gamma - \{\#\})^+) \cup (I \times Q \times \{\#\} \times Q \times (\Gamma - \{\#\})^* \{\#\})$ je množina pravidel, kde I je množinou všech kladných celých čísel.

Analogicky se zásobníkovým automatem, místo zápisu pravidla $(A, p, a, w, q) \in R$ používáme $Apa \to wq \in R$. *Konfigurace* automatu M je libovolný řetězec $\tau \in \Gamma^* Q \Sigma^*$.

Definice 2.22. Nechť $xApay, xwqy \in \Gamma^* Q \Sigma^*$ jsou konfigurace a $Apa \Rightarrow wq \in R$, pak *derivační krok* z $xApay$ do $xwqy$ je definován jako

$$xApay \Rightarrow xwqy.$$

Označení $\Rightarrow^k$ pro $k \geq 0$ vyjadřuje posloupnost k derivačních kroků. Není–li k shora omezené, pak zapisujeme $\Rightarrow^k *$. Platí–li navíc, že $k > 0$, pak používáme označení $\Rightarrow^+$.

Kapitola 3

Stavové gramatiky

Stavové gramatiky řádně popsal a definoval T. Kasai [4] v roce 1970. Koncept těchto gramatik spočívá ve speciálním stavovém mechanismu, jenž svým principem připomíná řízení stavů v konečných automatech. Každý stav stavové gramatiky totiž vymezuje pravidla, jenž mohou být v daném stavu použita. V každém derivačním kroce je přepsán takový nejlevější neterminál, pro který v daném stavu existuje alespoň jedno pravidlo gramatiky. V praxi je tedy možné aplikovat pravidla i na tzv. „zanořené neterminály“. Taková aplikace pravidla bývá označována za tzv. „zanořenou derivaci“. Během každého derivačního kroku dochází i ke změně stavu. Tím vzniká kontextová závislost, neboť každá aplikace pravidla zároveň ovlivní i výběr pravidla v dalším kroce.

3.1 Stavové gramatiky

Definice 3.1. Stavová gramatika G_k Kasaie [4] je pětice

$$G_k = (V_k, \Sigma_k, Q_k, R_k, S_k),$$

kde

- V_k je úplná abeceda
- $\Sigma_k \subset V_k$ je abeceda terminálů
- Q_k je konečná množina stavů
- $R_k \subset (Q_k \times (V_k - \Sigma_k)) \times (Q_k \times V_k^+)$ je konečná množina pravidel tvaru $(q, A, p, v) \in R_k$
- $S_k \in V_k - \Sigma_k$ je počáteční symbol

Pro zjednodušení a pro odstranění zbytečného nedeterminismu plynoucího z neznámého počátečního stavu (viz kapitola 4), rozšíříme definici stavových gramatik o počáteční stav.

Definice 3.2. Stavová gramatika G, rozšířená o počáteční symbol, je definována jako šestice

$$G = (V, \Sigma, Q, R, S, s),$$

kde:

- V, Σ, Q, R and S jsou definovány stejně jako V_k, Σ_k, Q_k, R_k resp. S_k v definici 3.1.

- $s \in Q$ je počáteční stav

Místo relačního zápisu pravidel $(q, A, p, v) \in R$, dále označovaných i jako přepisovací nebo také derivační pravidla, píšeme $(q, A) \to (p, v) \in R$.

Definice 3.3. Aktuální konfigurace K je u stavových gramatik definována jako trojice $K = (q, w, u)$, kde

- $q \in Q$ je aktuální stav
- $w \in V^*$ je aktuální celý derivační řetězec
- $u \in Q$ je aktuální vstupní řetězec

Definice 3.4. Nechť $w \in V^*$ je řetězec nad úplnou abecedou, který se může vzniknout během aplikování pravidel v průběhu analýzy, pak množina neterminálních symbolů v řetězci w, $nonterms(w)$, je definována jako:

$$nonterms(w) = \{n | n \in substring(w), n \in V - \Sigma\}$$

Definice 3.5. Nechť $w \in V^*$ je řetězec nad úplnou abecedou (dále jen derivační řetězec) a $q \in Q$ je aktuálním stavem, pak množina aplikovatelných pravidel na řetězev w ve stavu q, $rules(w, q)$, je definována jako:

$$rules(w, q) = \{(q, A) \to (p, v) \in R | A \ \in nonterms(w)\}$$

Nechť $x, y \in V^*$ jsou derivační řetězce, $q \in Q$ je aktuální stav, $rules(x, q) = \emptyset$, $rules(A, q) \neq \emptyset$ a pravidlo $\pi = (q, A) \to (p, v) \in R$, pak derivační řetězec xAy bude ve stavu q přepsán na derivační řetězec xvy aplikováním pravidla π gramatiky G. Tato operace se formálně zapisuje jako $(q, xAy) \Rightarrow (p, xvy)[\pi]$ nebo zjednodušeně $(q, xAy) \Rightarrow (p, xvy)$. Nechť $u \in V^*$ je derivační řetězec, pak gramatika G udělá nulový derivační krok z u do u, což zapisujeme jako $u \Rightarrow_0 u\ [\varepsilon]$ nebo zjednodušeně $u \Rightarrow_0 u$. Nechť $n \geq 1$, $u_0, \ldots, u_n \in V^*$ jsou derivační řetězce, $\pi_0, \ldots, \pi_n \in R$ jsou pravidly gramatiky G a pro každé $i = 0, \ldots, n$ existuje derivační krok $u_{i-1} \Rightarrow u_i[\pi_i]$, pak gramatika G udělá n derivačních kroků z u_0 do u_n, $u_0 \Rightarrow^n u_n[\pi_1 \ldots \pi_n]$ nebo zjednodušeně $u_0 \Rightarrow^n u_n$. Nechť $u_0 \Rightarrow^n u_n$ a $n \geq 0$, pak u_0 derivuje u_n v G, formálně zapsáno jako $u_0 \Rightarrow^* u_n$. Nechť $u_0 \Rightarrow^n u_n$ a $n \geq 1$, pak u_0 derivuje u_n v G, formálně zapsáno jako $u_0 \Rightarrow^+ u_n$.

Definice 3.6. Jazyk L popsaný stavovou gramatikou G, označovaný jako $L(G)$, je definován jako:

$$L(G) = \{w \in \Sigma^* | (q, S) \to (p, w) \in R,\ p, q \in Q\}$$

Příklad 3.1. Nechť $L = \{a^n b^n c^n | n \geq 0\}$ je jazyk popsaný stavovou gramatikou

$$G = (\{S, A, B, C, a, b, c\}, \{s, p_1, p_2, q_1, q_2, f\}, \{a, b, c\}, R, S, s)),$$

kde R obsahuje nsáledující pravidla:

$$\begin{array}{lll} (s, S) \to (s, ABC) & & \\ (s, A) \to (p_1, aA) & (p_1, B) \to (p_2, bB) & (p_2, C) \to (s, cC) \\ (s, A) \to (q_1, \varepsilon) & (q_1, B) \to (q_2, \varepsilon) & (q_2, C) \to (f, \varepsilon) \end{array}$$

Tyto pravidla jsou vypsány zcela záměrně tak, aby vynikla vzájemná spojitost mezi některými z nich. Na druhém a třetím řádku se totiž nacházejí pravidla, jenž budou při derivaci následovat vždy za sebou. Zároveň obě skupiny pravidel vycházejí ze stejného místa (konfigurace). První pravidla v obou skupinách aplikují pravidlo na neterminál A ve stavu s.

Na začátku derivace je tedy aplikací pravidla $(s, S) \to (s, ABC)$ přepsán neterminál S na derivační řetězec ABC. V nově vzniklé konfiguraci jsou aplikovatelná 2 pravidla a dochází ke výše zmíněnému větvení. V případě přijetí terminálu a aplikací pravidla $(s, A) \to (p_1, aA)$ přejde analýza do stavů p_1 a p_2 a vyprodukuje terminály b resp. c aplikací pravidel $(p_1, B) \to (p_2, bB)$ a $(p_2, C) \to (s, cC)$ a končí opět ve stavu s. V opačném případě je použito vymazávací pravidlo $(s, A) \to (q_1, \varepsilon)$, které bude následováno pravidly $(q_1, B) \to (q_2, \varepsilon)$ a $(q_2, C) \to (f, \varepsilon)$. Druhá větev se nevrací do stavu s, ale naopak končí analýzu.

Pokud tedy budeme chtít přijmout vstupní řetězec $aabbcc \in L$, pak bude průběh analýzy vypadat následovně:

$$\begin{aligned}
(s, S) &\Rightarrow (s, ABC) && [(s, S) \to (s, ABC)] \\
&\Rightarrow (p_1, aABC) && [(s, A) \to (p_1, aA)] \\
&\Rightarrow (p_2, aAbBC) && [(p_1, B) \to (p_2, bB)] \\
&\Rightarrow (s, aAbBcC) && [(p_2, C) \to (s, cC)] \\
&\Rightarrow (p_1, aaAbBcC) && [(s, A) \to (p_1, aA)] \\
&\Rightarrow (p_2, aaAbbBcC) && [(p_1, B) \to (p_2, bB)] \\
&\Rightarrow (s, aaAbbBccC) && [(p_2, C) \to (s, cC)] \\
&\Rightarrow (q_1, aabbBccC) && [(s, A) \to (q_1, \varepsilon)] \\
&\Rightarrow (q_2, aabbccC) && [(q_1, B) \to (q_2, \varepsilon)] \\
&\Rightarrow (f, aabbccc) && [(q_2, C) \to (f, \varepsilon)]
\end{aligned}$$

3.2 n-limitované stavové gramatiky

Uvažujme, že všechny pravidla gramatiky jsou aplikována na první až n-tý neterminál zleva, kde $n \geq 1$, pak o gramatice říkáme, že je n-limitovaná. Rozšířením pravidel o hloubku derivovaného neterminálu dostaneme gramatiku generující stejný jazyk s tím rozdílem, že proces syntaktické analýzy se výrazně zjednoduší. Formální definice n-limitovaných stavových gramatik s takovým rozšířením vypadá následovně:

Definice 3.7. n-limitovaná gramatika s pravidly, které mají určenou hloubku derivace, je sedmice

$$G_n = (D_n, V_n, \Sigma_n, Q_n, R_n, S_n, s_n),$$

kde:

- D_n je množina přirozených čísel i, kde $1 \leq i \leq n$
- V_n je úplná abeceda
- $\Sigma_n \subset V_n$ abeceda terminálů
- Q_n je konečná množina stavů

- $R_n \subset D_n \times (Q_n \times (V_n - \Sigma_n)) \times (Q_n \times V_n^+)$ konečná množina pravidel tvaru $(N, p, A) \to (q, w)$
- $S_n \in V_n - \Sigma_n$ je počáteční neterminál
- $s_n \in Q_n$ je počáteční stav

Derivační krok v G_n je zapisován jako $(q, xAy) \ {}_n\Rightarrow \ (p, xvy)[\pi]$ nebo zjednodušeně $(q, xAy) \ {}_n\Rightarrow \ (p, xvy)$. Sekvence derivačních kroků se také podstatně neliší. Nechť existuje $m \geq 0$ a $v, w \in W \times V^+$, pak pro vyjádření n-limity sekvencí derivačních kroků $v \Rightarrow^m w$, $v \Rightarrow^* w$ a $v \Rightarrow^+ w$ používáme zápis $v \ {}_n\Rightarrow^m w$, $v \ {}_n\Rightarrow^* w$ resp. $v \ {}_n\Rightarrow^+ w$.

3.3 Generativní síla

Nechť $\mathbf{ST}$, $\mathbf{ST^{Erasing}}$, $\mathbf{CF}$, $\mathbf{CS}$, $\mathbf{RE}$ jsou po řadě označení pro rodiny jazyků generované stavovými gramatikami, stavovými gramatikami s ε-pravidly, bezkontextovými gramatikami, kontextovými gramatikami a neomezenými gramatikami.

Pro každé $n \geq 1$ definujeme $\mathbf{ST_{n-Lim}}$ a $\mathbf{ST^{Erasing}_{n-Lim}}$ reprezentující rodiny jazyků generované n-limitovanými stavovými gramatikami resp. n-limitovanými stavovými gramatikami s ε-pravidly. Rodiny jazyků generované všemi n-limitovanými stavovými gramatikami a všemi n-limitovanými stavovými gramatikami s ε-pravidly, po řadě označované jako $\mathbf{ST}_\infty$ resp. $\mathbf{ST}_\infty^{\mathbf{Erasing}}$, jsou definovány následovně:

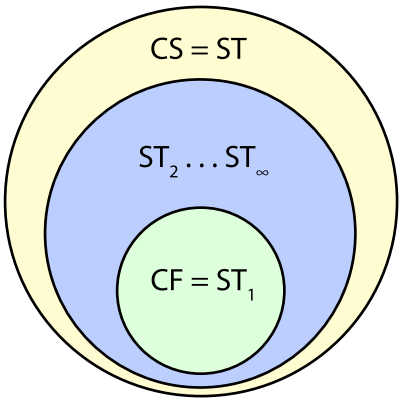
$$\mathbf{ST}_\infty = \bigcup_{n \geq 1} \mathbf{ST_n}$$

$$\mathbf{ST}_\infty^{\mathbf{Erasing}} = \bigcup_{n \geq 1} \mathbf{ST_n^{Erasing}}$$

Teorém 3.8. $\mathbf{CF} = \mathbf{ST_1} \subset \mathbf{ST_2} \subset \cdots \subset \mathbf{ST}_\infty \subset \mathbf{ST} = \mathbf{CS}$.

Teorém 3.9. *Všechny* $\mathbf{ST_{n-Lim}}$ *pro* $n \geq 1$ *reprezentují abstraktní rodinu jazyků.*

Teorémy 3.8 a 3.9 dokázal Kasai [4].



Obrázek 3.1: Zařazení stavových gramatik v rámci Chomského hierarchie.

Teorém 3.10. $\mathbf{ST^{Erasing}} = \mathbf{RE}$

Důkaz teorému 3.10 popsal Meduna [2].

Teorém 3.11. $\mathbf{CF} = \mathbf{ST_1^{Erasing}}$

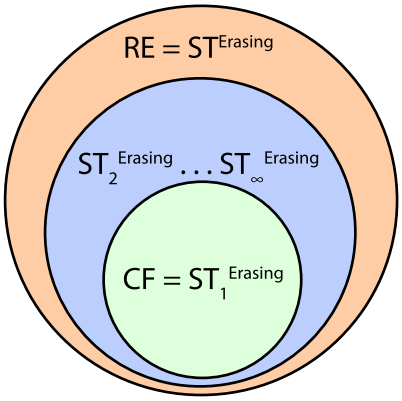
Důkaz 3.12. Z teorému 3.8 vyplývá, že pro $n = 1$ je generativní síla stejná jako u bezkontextových gramatik. Pro každou gramatiku v $\mathbf{ST_1}$ tedy existuje ekvivalentní **CF**. Vytvoříme–li transformaci mezi **CF** a $\mathbf{ST_1}$ takovou, že bude obsahovat pravidlo, které dvojice na pravé straně (stav a neterminál) v $\mathbf{ST_1}$ nahradí novým neterminálem v **CF**, pak nebude rozdíl mezi vymazávacími pravidly obou těchto gramatik. Obě gramatiky tedy generují stejnou rodinu jazyků.

Teorém 3.13. $\mathbf{CF} = \mathbf{ST}_1^{Erasing} \subset \mathbf{ST}_2^{Erasing} \subset \cdots \subset \mathbf{ST_\infty^{Erasing}} \subset \mathbf{ST^{Erasing}} = \mathbf{RE}$

Důkaz 3.14. Důkaz přímo vyplývá z teorémů 3.8, 3.10 a 3.11.

Teorém 3.15. *Všechny* $\mathbf{ST_{n-Lim}^{Erasing}}$ *pro* $n \geq 1$ *reprezentují abstraktní rodinu jazyků.*

Důkaz 3.16. Důkaz přímo vyplývá z teorémů 3.8, 3.9 a 3.13.

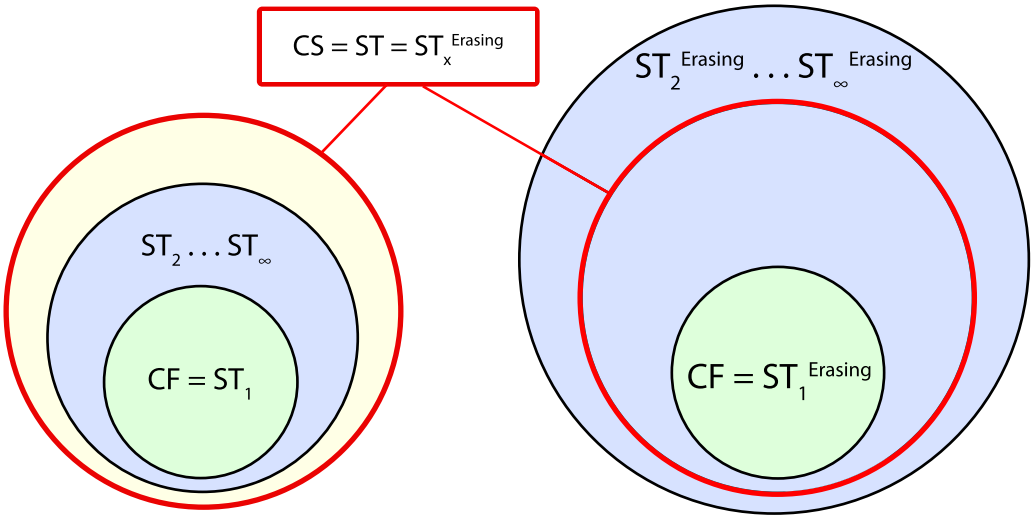


Obrázek 3.2: Zařazení stavových gramatik s ε-pravidly v rámci Chomského hierarchie.

Teorém 3.17. $\mathbf{ST_{n-Lim}} \subseteq \mathbf{ST_{n-Lim}^{Erasing}}$ *pro* $n \geq 1$ [5].

Domněnka 3.18. $\exists x > 1$ takové, že $\mathbf{ST_x^{Erasing}} = \mathbf{CS}$.

Tato domněnka se potvrdí např. dokázáním, že platí vztah $\mathbf{ST_{y+1}} \subseteq \mathbf{ST_y^{Erasing}}$ pro $y > 1$. Následnou inkluzí by pak bylo prokázáno, že s každou inkrementací limity stavových gramatik vzrůstá i rozdíl v generativní síle n-limitovaných stavových gramatik oproti gramatikám stejné limity ovšem s povolenými vymazávacími pravidly. Předpokládanou situaci demonstruje obrázek 3.3.



Obrázek 3.3: Předpokládaný vzájemný vztah generativní síly $\mathbf{ST^{Erasing}}$ a **CS**.

Kapitola 4

Nedeterminismus ve stavových gramatikách

Tato kapitola se zaměřuje na nedeterminismus v analýze stavových gramatik. Nejprve jsou zde popsány základní nejednoznačnosti výběru z více aplikovatelných pravidel způsobené zavedením stavového mechanismu a zanořené derivace. Sekce [x] se zaměřuje a nedeterminismus způsobený povolením ε–pravidel a to zejména v zanořené derivaci. Závěrečná sekce této kapitoly se zaměřuje na nejobsáhlejší problém, kterým je rekurze.

4.1 Základní nedeterministické situace

Tato sekce popisuje nejednoznačnost výběru pravidel a vliv stavů a hloubky zanoření na výběr pravidla. Většina z těchto nedeterministických problémů vychází z nejednoznačných výběrů pravidel v bezkontextových gramatikách, nicméně u stavových gramatik jsou tyto problémy rozšířeny navíc o změnu stavu a tedy i možnou změnu kontextu. Z tohoto důvodu zde budou uvedeny právě i známé nedeterministické situace, ze kterých vychází řešení pro stavové gramatiky.

Tento problém demonstruje důvod zavedení definice 3.2. Mějme tedy gramatiku G_1, která obsahuje počáteční neterminál S a následující pravidla:

$$(p, S) \to (p, aw), \tag{4.1}$$
$$(q, S) \to (q, az), \tag{4.2}$$

kde $w, z \in V^*$. V případě použití původní definice stavových gramatik (viz definice 3.1), tak už při úplně první derivaci nastává problém. Není znám počáteční stav, a proto je nutné prohledat všechny pravidla všech stavů, abychom našli všechny, jenž derivují počáteční neterminál. Tomuto problému lze předejít vynucením počátečního stavu, tedy jako v definici 3.2, a to tak, že přidáním nového stavu $u \notin Q$ a duplikací všech dříve nalezených pravidel následujícím způsobem:

$$(u, S) \to (q, aw), \tag{4.3}$$
$$(u, S) \to (p, av), \tag{4.4}$$
$$(p, S) \to (p, aw), \tag{4.5}$$
$$(q, S) \to (q, av). \tag{4.6}$$

Všechna nová pravidla se liší od původních svým počátečním stavem u. Toto řešení ještě není úplné, chybí totiž vyřešit konflitní prefixy, což je dalším ukázkovým případem nedeterminismu. Mějme tedy první 2 pravidla z předchozího řešení

$$(u, S) \to (q, aw), \tag{4.7}$$
$$(u, S) \to (p, av). \tag{4.8}$$

Tyto pravidla již mají shodnou levou stranu, tedy stav a derivovatelný neterminál, a mají shodný prefix derivačního řetězce. Liší se však v cílovém stavu, ale ten v tomto konkrétním případě nehraje žádnou roli. Této situaci lze opět předejít a to vytknutím shodného prefixu do pravidla $(u, S) \to (u, a)$. Je však nutné zajistit, aby další derivační krok vedl do jednoho z původních pravidel. Zajistit takového chování lze docílit přidáním nového neterminálu $X \notin V$ a nahrazením původních pravidel novými. Výsledná pravidla pak budou vypadat následovně:

$$(u, S) \to (p, aX), \tag{4.9}$$
$$(u, X) \to (p, w), \tag{4.10}$$
$$(u, X) \to (q, v). \tag{4.11}$$

Terminál není jediným symbolem, který se může objevit na začátku derivačního řetězce. Jedno či více pravidel může mít neterminálový prefix jako v následující trojici pravidel

$$(u, S) \to (p, Xw), \tag{4.12}$$
$$(u, S) \to (q, Yz, \tag{4.13}$$
$$(u, S) \to (f, y), \tag{4.14}$$

kde XY jsou neterminály, $w, z \in V^*$ jsou řetězce a $y \in \Sigma$ je symbol abecedy. Pro takové pravidla existují v bezkontextových gramatikách různé mechanismy, které umožňují určit, které pravidlo pro daný vstupní symbol použít. Jedním z těchto způsobů je tvorba tzv. „LL-tabulky“ [6]. Tyto mechanismy zpravidla fungují na principu předderivování jednotlivých pravidel. Pokud budeme chtít tento princip zopakovat i u stavových gramatik, je nutné vzít v potaz změnu pravidel a hloubku derivace. Pro nejlevější derivaci se nic nemění, ale v případě zanořené derivace může nastat problém. Tento problém demonstruje následující skupina pravidel

$$(s, S) \to (p, AB), \tag{4.15}$$
$$(p, B) \to (q, X, \tag{4.16}$$
$$(q, X) \to (q, b, \tag{4.17}$$
$$(q, A) \to (f, a), \tag{4.18}$$
$$(f, B) \to (f, c). \tag{4.19}$$

Pravidlem 4.15 uvozujeme stav aktuálního derivačního řetězce. V cílovém stavu p tohoto pravidla je možné jako další krok použít pravidlo 4.16 s tím, že aplikací dalšího pravidla 4.17 budeme očekávat následující výslednou změnu $AB \Rightarrow Ab$. K tomu však nedojde, neboť

po aplikaci pravidla 4.16 bude následovat pravidlo 4.18, protože neterminál A je vlevo od neterminálu B. Po aplikaci 4.18 a následně 4.19 tedy bude derivační řetězev ac. Z toho vyplývá, že rozderivování pravidel stavové gramatiky je nutné řešit v kontextu a to od počátečních pravidel až po aplikaci požadovaného pravidla, nikoliv zcela nezávisle jako u bezkontextových gramatik.

4.2 ε–pravidla v zanořené derivaci

Speciálním případem jsou vymázávací pravidla s různým cílovým stavem. Výběr pravidla je mnohem komplikovanější, neboť zde neexistuje žádný záchytný bod v podobě symbolů derivačního řetězce, který je zde prázdný. Jako příklad problému mějme následující pravidla gramatiky:

$$(s, S) \to (q, \varepsilon) \tag{4.20}$$
$$(s, S) \to (p, \varepsilon) \tag{4.21}$$

Inspirací k alespoň částečnému řešení mi posloužily LL gramatiky [6], přesněji množiny *first()*, *empty()*, *follow()* a *predict()*. Algoritmy tvorby těchto množin nedělají nic jiného, než že postupně rozderovávají pravidla a na základě možných derivačních řetězců určují aplikovatelné pravidla. Přesto to stejné je možné i u stavových gramatik, ale jen v omezené míře. Přesněji jen pro pravidla, jejichž cílový stav je stejný, jako výchozí stav v levé části pravidla, nebo v případě, kdy je derivován právě nejlevější neterminální symbol. V žádném z těchto 2 případů totiž nehrozí riziko nežádoucí změny kontextu. Další vysvětlení lépe objasní postupná aplikace následujích pravidel:

$$\begin{aligned}(s, S) &\to (q, ABC)\\ (q, B) &\to (p, \varepsilon)\\ (p, C) &\to (f, c)\\ (p, A) &\to (u, X)\\ (u, X) &\to (y, w)\\ (q, B) &\to (f, \varepsilon)\\ (f, C) &\to (x, X)\\ (x, A) &\to (s, c)\end{aligned}$$

a aktuální konfigurací (s, S, c). V dané konfiguraci lze aplikovat první pravidlo, jenž vygeneruje derivační řetězec ABC. Zde ale vzniklá výše popsaný problém výběru pravidla. V případě využití postupu tvorby množin z LL gramatik se jeví aplikace pravidla $(q, B) \to (p, \varepsilon)$ správná, neboť neterminál C přímo následující za aktuálně derivovaným neterminálem B derivuje řetězec začinající symbolem c, jenž se nachází i na vstupu testované věty. Nicméně provedení následujících kroků:

$$\begin{aligned}(s, S) &\Rightarrow (s, ABC) && [(s, S) \to (q, ABC)]\\ &\Rightarrow (p, AC) && [(q, B) \to (p, \varepsilon)]\\ &\Rightarrow (u, XC) && [(p, A) \to (u, X)]\\ &\Rightarrow (y, wC) && [(u, X) \to (y, w)]\end{aligned}$$

ukazuje, že k derivaci neterminálu C nikdy dojít nemůže, neboť dojde k aplikaci v daný ukamžik levějšího neterminálu, jenž posune analýzu jiným směrem. Aplikace předchozího vymazávacího pravidla se tedy stává bezdůvodná. Na druhou stranu aplikace pravidla $(q, B) \to (f, \varepsilon)$, které zdánlivě nemůže přijmout vstupní symbol c, nakonec vede k jeho přijetí:

$$\begin{aligned} (s, S) &\Rightarrow (s, ABC) & [(s, S) &\to (q, ABC)] \\ &\Rightarrow (f, AC) & [(q, B) &\to (f, \varepsilon)] \\ &\Rightarrow (x, AX) & [(f, C) &\to (x, X)] \\ &\Rightarrow (s, cX) & [(x, A) &\to (s, c)] \end{aligned}$$

Tento problém u nejlevější derivace nenastane, jedná se o problém přímo spojený se zanořenou derivací, který způsobuje změnu kontextu.

4.3 Levá, pravá a kombinovaná rekurze

Levou rekurzí jsou myšlena pravidla typu $(p, A) \to (p, Aw)$, kde $w \in V^*$. Jedná se o nekonečnou smyčku, která jen generuje, ale sama se nedokáže ukončit. Uvedené pravidlo představuje triviální levou rekurzi, kterou lze odstranit prohozením prvního neterminálu v derivačním řetězci se zbylou částí (suffixem) téhož řetězce a to pouze v případě, že takto vznikne nové pravidlo. V opačném případě je pravidlo zbytečné a je nutné ho odstranit. Příklady jednoduché levé rekurze a jejího řešení:

$$\begin{aligned} (p, A) &\to (p, AwB) & (p, A) &\to (p, wBA) \\ (p, A) &\to (p, Aw) & (p, A) &\to (p, wA) \\ (p, A) &\to (p, A) & & nelze \end{aligned}$$

Jelikož stavové gramatiky umožňují derivaci zanořeného neterminálu, je potřeba kontrolovat i případnou pravou rekurzi. Ta může vzniknout v situaci, kdy v konfiguraci suB, kde $u \in \Gamma^+$ a $rules(u, s) = \emptyset$ provádíme derivační krok tvaru

$$(s, B) \to (s, wBv), \tag{4.22}$$

kde $w, v \in \Gamma^*$ a $rules(w, s) = \emptyset$.

Existují však i mnohem komplikovanější rekurze, které se skládají z posloupností více pravidel a to zejména tehdy, když se v rámci těchto posloupnosti pravidel mění stav.

$$(p, A) \to (p, B) \tag{4.23}$$
$$(p, B) \to (p, Aw) \tag{4.24}$$

Je–li ale během této posloupnosti pravidel možné aplikovat více pravidel se stejnou levou stranou pravidla, a obzvlášť pokud tyto pravidla lze aplikovat právě na nejlevější neterminál, tak může (ale nemusí) existovat mechanismus zastavení této rekurze. Příkladem je následující dvojice pravidel

$$(p, A) \rightarrow (p, Aa) \tag{4.25}$$
$$(p, A) \rightarrow (p, \varepsilon) \tag{4.26}$$

Tato posloupnost je zastavena pravidlem 4.26. Zatímcto následující dvojice ukončující podmínku nemá

$$(p, A) \rightarrow (p, Aa) \tag{4.27}$$
$$(p, A) \rightarrow (p, Ab) \tag{4.28}$$

Nejkomplexnější příkladem rekurze je pak kombinovaná rekurze, která se skládá z pravidel, které nejenže mění stavy, ale také procházející různými hloubkami derivace. Příkladem může být následující posloupnost pravidel

$$(s, S) \rightarrow (p, ASB) \tag{4.29}$$
$$(p, B) \rightarrow (q, Bw) \tag{4.30}$$
$$(q, A) \rightarrow (s, vA) \tag{4.31}$$

Taková rekurze je kontextově závislá a je jen obtížně detekovatelná při analýze gramatiky. Obecně lze říct, že problém rekurze je jediným destruktivním prvkem analýzy. Rekurzi však lze detekovat v průběhu analýzy v závislosti na jejím konkrétním typu, a to právě když

1. byla provedena sekvence derivačních kroků, během které nebyl přijat žádný vstupní symbol
2. došlo k opakovanému použití stejného pravidla
3. počet výskytů derivovaného neterminálu není menší než při předchozí aplikaci téhož pravidla

Kapitola 5

Návrh metody syntaktické analýzy

Tato kapitola popisuje návrh metody syntaktické analýzy shora–dolů založené na stavových gramatikách. Cílem je navrhnout takový analyzátor, který bude schopný zohlední všechny nedeterministické situace popsané v kapitole 4. První sekce této kapitoly shrnuje veškeré požadavky na vlastnosti metody, nastiňuje možné koncepty řešení a hodnotí jejich předpokládané vlastnosti. Sekce 5.2 popisuje zvolený koncept sekvenční analýzy s aplikací více pravidel a s návratem. Sekce 5.3 navazuje na koncept sekvenční návrhu a detailněji popisuje jeho paralelní modifikaci. V této kapitole jsou také řešení problémy detekce a odtranění rekurze (viz sekce 5.4), dále vliv vhodné transformace gramatiky na počet větvení během analýzy (viz sekce 5.5). Předposlední sekce 5.6 shrnuje problematiku nalezení skutečné chyby, jež způsobila zamítnutí vstupního řetězce. V závěru kapitoly jsou pak shrnuty vlastnosti navržené metody.

5.1 Koncept analýzy

Základní myšlenka: Nástrojem pro rozpoznávání jazyků generovaných stavovými gramatikami je hluboký zásobníkový automat. Tento automat je však v principu nedeterministický, a proto je třeba jej buď modifikovat nebo jeho činnost podpořit dalším nástrojem. Před vyřešením tohoto dilema je vhodné připomenout, že problém nedeterminismu ve stavových gramatikách je dvojího typu: bezkontextový, který lze řešit v předstihu patřičnou úpravou gramatiky, a kontextový, který v předstihu úpravou na stejný typ gramatiky nelze. Je však možné použít jiný formalismus pro popis gramatiky, která není bezkontextová. Takovým je např. použití kontextových pravidel. Použití takového formalismu s sebou přináší problémy při návrhu gramatiky, jelikož kontextová pravidla nejsou intuitivní.

Další možností je předzpracování pravidel s cílem vytvořit tabulku pro aplikování jednotlivých pravidel v konkrétních stavech a patřičném výskytu neterminálů tak, jako je tomu u LL gramatik [8]. Zde však situaci komplikuje možnost narušení posloupnosti pravidel vedoucí k přijetí konkrétního terminálu jiným pravidlem v levějším kontextu (viz sekce 4.2). Bylo by tedy nutné předgenerovat všechny možné řetězce daného jazyka nebo provést transformaci gramatiky tak, abychom dostali zbývající údaje. Takový postup však nemusí u stavových gramatik vést ke konečnému řešení a to zejména kvůli nekonečnosti daného jazyka resp. doposud nenalezené ekvivalentní úpravy pravidel pro odstranění rekurze.

Zbývá tedy přesunout problém výběru pravidla až do samotné analýzy. Tím sice můžeme sloučit řešení problému aplikace více možných pravidel s detekcí rekurze, nicméně jen za cenu zdlouhavější analýzy. V takovém případě se v každém kroce analýzy kontroluje množ-

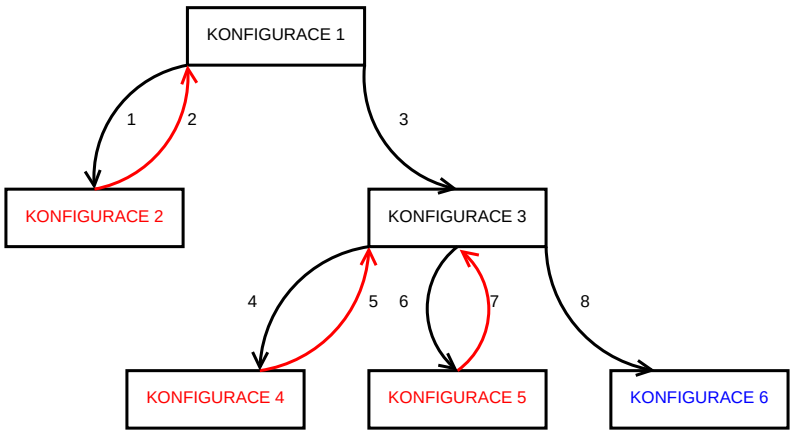
ství aplikovaných pravidel. V případě většího počtu takových pravidel je z nich vždy jedno vyjmuto a dále zkoumáno (tj. dochází k dohledání a případné aplikaci dalších pravidel). Je–li zjištěno, že aplikace daného pravidla vede k zamítnutí vstupního řetězce, tak pak je zkoumáno další z možných pravidel. Takový koncept analýzy lze abstrahovat na problém prohledávání stavového prostoru [11, 10], kde stavy v tomto případě představují konfigurace vzniklé aplikací jednotlivých pravidel. Pro řešení takového problému existují algoritmy jako *prohledávání do hloubky*, *prohledávání do šířky* nebo třeba *metoda se zpětným návratem* (dále jen *backtracking*) [11, 3].

Jako koncept analýzy jsem zvolil poslední možnost, tedy *backtracking*, a to z důvodu, že se jedná o úplné a hlavně intuitivní prohledání všech možných posloupností derivačních kroků. Dalším důvodem je také skutečnost, že v takové analýze spatřuji mnohé možnosti pro redukci počtu větvení a obecně další optimalizaci (viz níže).

5.2 Sekvenční analýza s návratem

Základním výpočetním modelem pro práci se stavovými gramatikami je hluboký zásobníkový automat, tedy konečný automat rozšířený o zásobník a schopnost expandovat neterminální symbol v libovolné hloubce. Při použití prohledávací metody *backtracking* však budeme potřebovat ještě jeden zásobník, na který si budeme ukládat informace potřebné k obnově předchozího stavu analýzy. Ten je charakterizován aktuální konfigurací. Abychom se vyhnuli redundantním operacím spojeným s návratem v analýze (např. opakovaná lexikální analýza stejného úseku vstupu), osamostatníme syntaktickou analýzy ze zbylé části překladu. Ve výsledku tedy budeme očekávat, že na vstupu bude posloupnosti tzv. „tokenů“ (nebo jednoduše posloupnost terminálů) a naopak na výstupu bude posloupnost pravidel vedoucí k přejetí vstupního řetězce nebo chyba a určení její nejpravděpodobnější příčiny. Z tohoto důvodu je potřeba ukládat na pomocný zásobník dvojici (χ, τ), kde τ je nově vzniklá konfigurace a χ je uspořádaná posloupnost pravidel vedoucí od začátku analýzy až ke konfiguraci τ.

Činnost analýzy se v každém kroce skládá z analýzy stavu (přijetí terminálu nebo dohledání množiny aplikovatelných pravidel) a odpovídající akci metody backtracking. Jak funguje tato metoda demonstruje obrázek 5.1.



Obrázek 5.1: Jednotlivé kroky průchodu stromem analýzy metodou backtracking.

Obrázek 5.1 je velmi zjednodušující. Jednotlivé přechody mezi konfiguracemi mohou

skrývat více než jen jeden derivační krok. Červené přechody vznikají v momentě, kdy syntaktická analýza v dané větvi tohoto stavového prostoru zamítne vstupní posloupnost, čímž si vynutí návrat. Není–li se kam vrátit (zásobník s konfiguracemi je prázdný), pak je syntaktická analýza prohlášena za neúspěšnou.

5.2.1 Syntaktická analýza v rámci jedné větve

Základní myšlenka: Tato část je jádrem celé analýzy. Obecně se jedná o konečný automat, který, zjednodušeně řečeno, v nekonečné smyčce rozhoduje o následujících operacích v rámci analýzy. V rámci každého cyklu je nutné nejprve zkontrolovat stav zásobníku (není–li uvedeno jinak, je pod pojmem zásobník myšlen zásobník patřící k automatu). V případě, že je zásobník prázdný, resp. obsahuje pouze speciální symbol #, znamená to ukončení analýzy dané větve. Je–li navíc vstupní řetězec také prázdný, jedná o konec úspěšný.

Pokud se na zásobníku nachází alespoň jeden další symbol, je nutné prověřit vrchol tohoto zásobníku. Je–li tímto symbolem terminál, ověříme, zda se shoduje s prvním symbolem na vstupu a případně jej přijmeme. Je–li na vrcholu zásobníku neterminál, pak je potřeba dohledat všechna pravidla, jenž lze použít, a poté všechny pravidla odděleně aplikujeme na stávající konfiguraci. Nelze–li použít žádné pravidlo, ukončíme analýzu dané větve jako neúspěšnou. Tento slovní popis je mnohem výstižněji zobrazen v diagramu aktivit[1] na obrázku 5.2.

5.2.2 Mechanismus větvení a návratu

Na obrázku 5.2 se nachází i jeden speciální blok „aplikace pravidel“. Tento úsek abstrahuje část logiky, o kterou se stará backtracking – ukládání na zásobník konfigurací. Je nutné, aby se na tomto zásobníku objevily pouze takové konfigurace, které vzniknou přímou derivací jednoho z pravidel, jenž lze v aktuálním stavu aplikovat, a nedošlo ke sloučení více pravidel. Příklad realizace ukládání je na obrázku 5.3.

Zde jsou nejprve vloženy na zásobník úplně všechny vzniklé konfigurace a až pak je z nich jedna odebrána a načtena. K načtení nové konfigurace dochází i v případě výskytu chyby v analýze, např. v obrázku 5.2 se jedná o blok „chyba“.

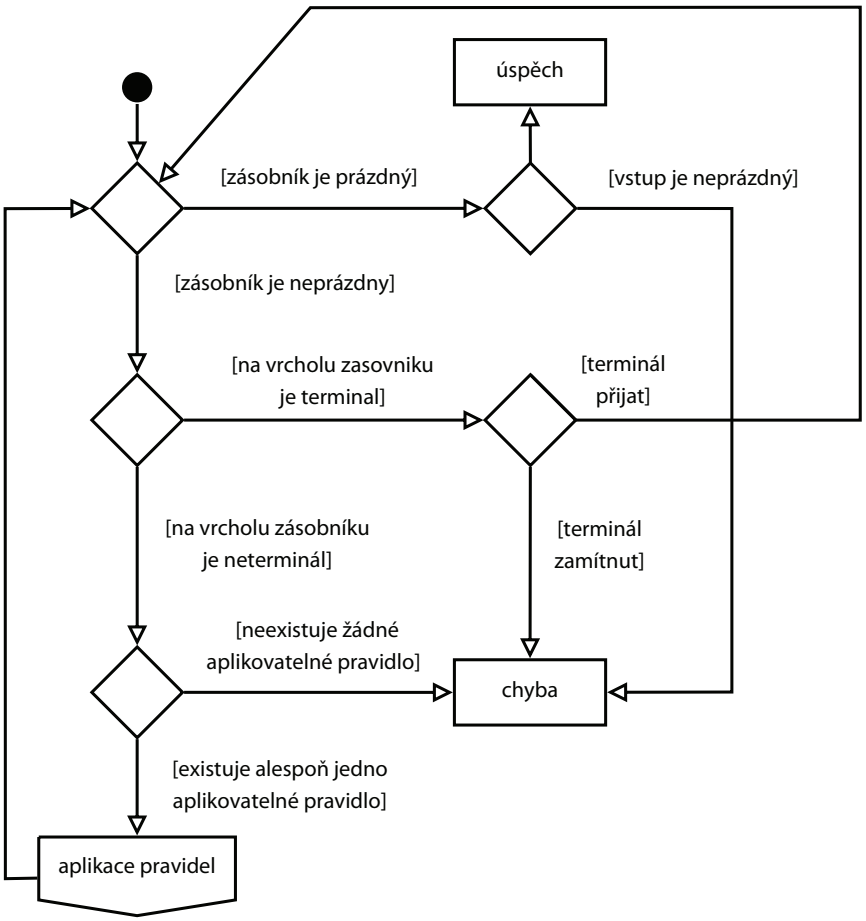
5.3 Paralelní analýza

Použitím metody backtracking lze dosáhnout úspěšného přijetí vstupního řetězce (ošetříme–li korektně případnou rekurzi viz níže), nicméně počet nutných kroků je v nejlepším možném případě roven počtu aplikovaných pravidel. Může se však stát (a v případě výskytu chyby se také stane), že se během analýzy touto metodou projde úplně celým stromem tvořeným všemi možnými konfiguracemi, což je zejména při sekvenční analýze poměrně hodně časově náročná operace. Z tohoto důvodu jsem koncept této sekvenční analýzy upravil tak, aby se mohla provádět syntaktická analýza ve více větvích paralelně viz obrázek 5.4.

V případě paralelního zpracování jsou tedy mechanismy ukládání a načtení konfigurace nahrazeny po řadě větvením analýzy resp. správcem větví. Úkolem takového správce je

- inicializace kořenové větve analýzy,

[1] Diagram aktivit je jedním z UML diagramů, které popisují chování. Tento diagram se používá pro modelování procedurální logiky a procesů [1]. Popis významu jednotlivých použitých bloků naleznete v příloze C.



Obrázek 5.2: Diagram aktivit popisující činnost hlubokého zásobníkového automatu.

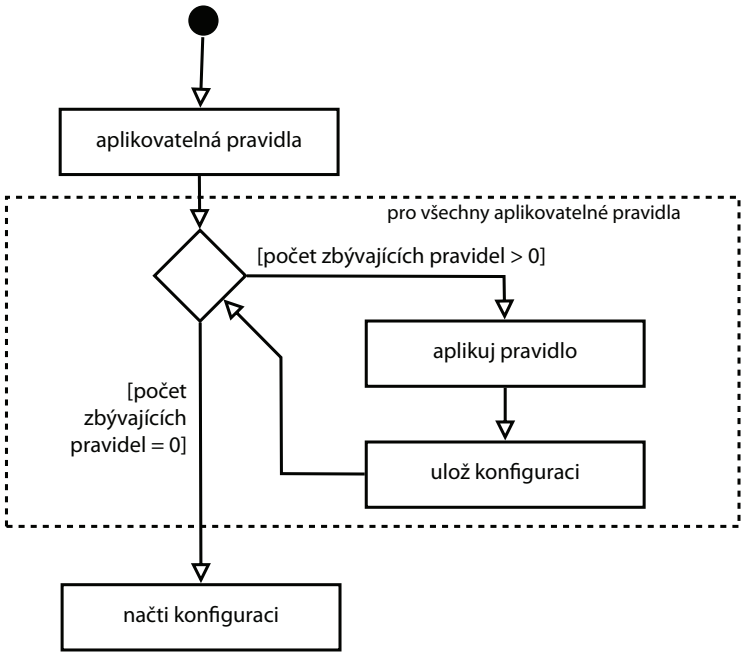
- korektní zpracování chyb z ukončených spelých větví,
- ukončení všech aktivních větví analýzy (tj. stále běžících procesů) v případě, kdy alespoň jedna větev ukončí svoji činnost přijetím vstupního řetězce.

Aplikace pravidel v paralelní analýze taky dozná určitých změn. Tou hlavní je oddělení prvního aplikovatelného pravidla od těch zbývajích. Ty totiž budou analyzovány ve zcela nových větvích analýzy, zatímco v současné větvi bude aplikováno právě to jedno oddělené pravidlo. Schéma aplikace pravidel je demonstrováno na obrázku 5.5.

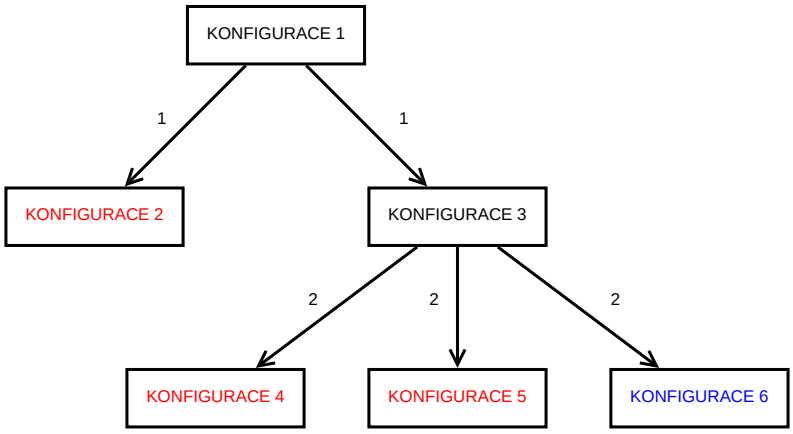
5.4 Detekce a odstranění rekurze

Z popisu rekurze (viz 4.3) vyplývá, že pro její detekci je potřeba uchovávat posloupnost n posledních aplikovaných pravidel, během kterých nebyl v analýze přijat žádný vstupní symbol. Zároveň je nutné uchovávat také jednotlivé derivační řětězce vzniklé po aplikaci těchto pravidel.

V případě splnění předchozích podmínek a zároveň pokud bylo poslední pravidlo aplikováno na nejlevější neterminál N, tak poté se jedná jen a pouze o levou rekurzi. Naopak



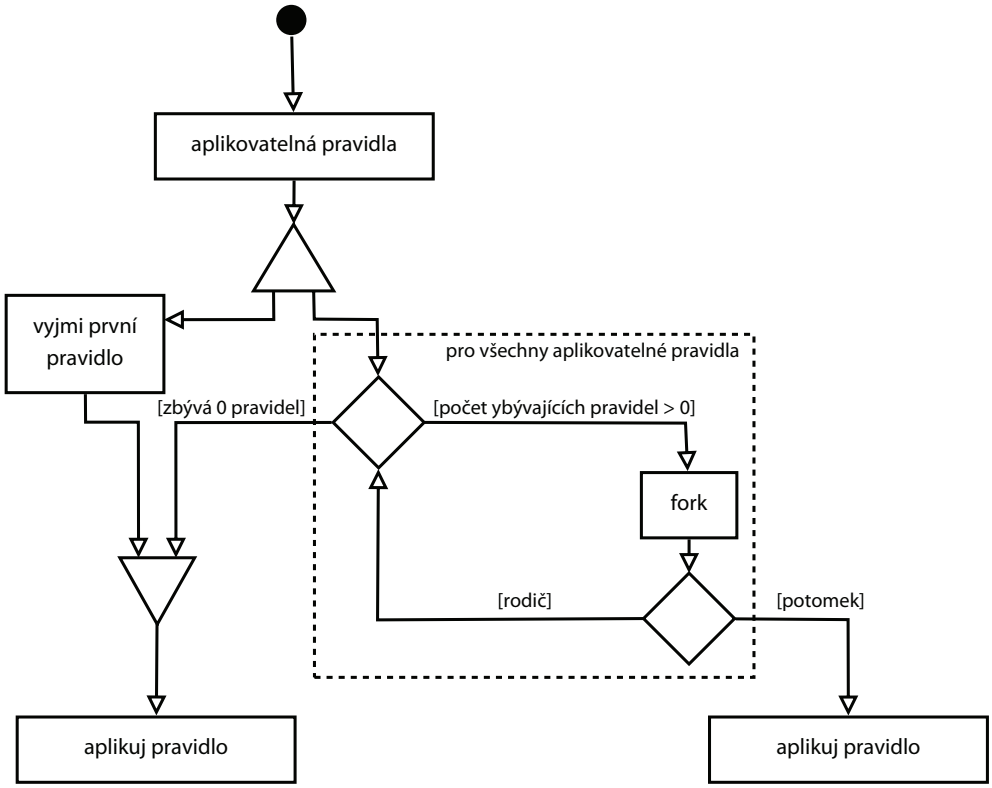
Obrázek 5.3: Mechanismus výběru z aplikovatelných pravidel a uložení zbývajících pravidel na zásobník konfigurací.



Obrázek 5.4: Jednotlivé kroky průchodu stromem analýzy při paralelním zpracování.

v případě, kdy je opakované pravidlo aplikováno na zanořený neterminál N, pak se pravděpodobně jedná o rekurzi libovolného typu. Ještě je však nutné nejprve ověřit, zda počet neterminálů N v aktuálním derivačním řetězci je větší nebo roven počtu N v derivačním řetězcích vzniklých po předchozí aplikaci téhož pravidla, a to od posledního přijetí vstupního symbolu. Pokud se počet N snižuje s každou aplikací téhož pravidla, pak se nejedná o rekurzi.

V případě, že budeme chtít rozlišit pravou popř. kombinovanou rekurzi od levé rekurze, stačí porovnat délku prefixů jednotlivých derivačních řetězců. Nechť uNw a xNy, kde $u, w, x, y \in \Gamma$, jsou po řadě 2 derivační řetězce vzniklé aplikací téhož pravidla, pak se jedná



Obrázek 5.5: Mechanismus paralelní aplikace všech pravidel.

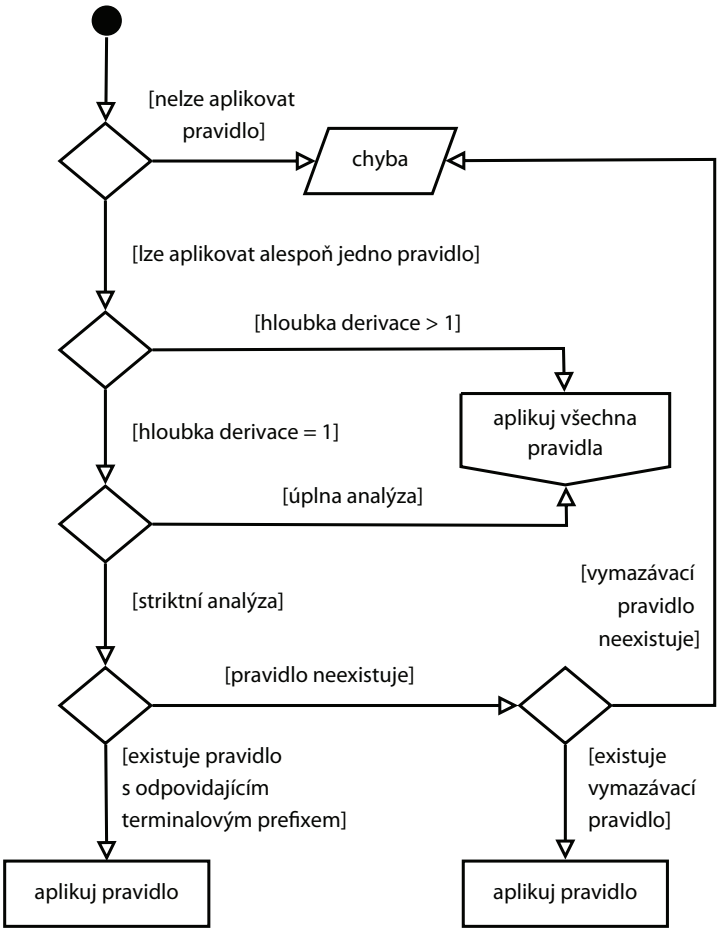
o

- levou rekurzi, pokud platí $|u| = |x|$,
- pravou rekurzi, pokud platí $|u| < |x|$ a zároveň $|w| = |y|$ nebo o
- kombinovanou rekurzi, neplatí–li předchozí 2 případy

5.5 Redukce počtu větvení při vhodné úpravě gramatiky

Základní myšlenka: Uvažujme gramatiku, která nebude obsahovat bezkontextový nedeterminismus v pravidlech, pak aplikace některých druhů pravidel při nejlevější derivaci bude zbytečná. Zavedeme–li navíc striktní mechanismus takový, že při nejlevější derivaci bude možné aplikovat pouze vymazávací pravidla nebo pravidla, jejiž derivační řetězec začíná terminálem, pak při nejlevější derivaci dojde k aplikaci nejvýše jednoho pravidla.

Na obrázku 5.6 je zobrazen způsob, jakým by byly aplikovány pravidla v případě vhodně upravené gramatiky. Této úpravy lze dosáhnout vytýkáním a spojováním pravidel viz 4. To samo o sobě však nestačí, neboť stále může dojít k situaci, kdy je možné aplikovat pravidlo s neterminálovým prefixem, který již není možné v cílovém stavu pravidla derivovat (je zde předpoklad, že dalším derivačním krokem bude aplikace pravidla na zanořený neterminál). Tento případ nedeterminismu však považuji u nejlevější derivace za zbytečný, neboť se domnívám, že ho lze nahradit vymazávacím pravidlem případne pravidlem s terminálovým prefixem. Proto tento případ užití bude v takové gramatice zakázán. Jako poslední možný



Obrázek 5.6: Úprava aplikace pravidel u vhodně upravené gramatiky.

nedeterministický problém tedy zůstává rekurze. V případě nejlevější derivace se jedná pouze o levou rekurzi, kterou lze převést na pravou rekurzi viz 4.3.

5.6 Určení skutečného výskytu chyby

V případech, kdy je analyzován nevalidní vstupní řetězec, bývá vhodné přesně specifikovat důvody jeho zamítnutí a tím zjednodušit případnou opravu. V navržené syntaktické analýze však vzniká problém, neboť její větvení způsobuje výskyt většího počtu možných chyb, jenže ne všechny jsou doopravdy způsobené nevalidním vstupním řetězcem. Některé větve jsou tzv. „slepé“, tedy nevedou pro daný vstup k úspěšnému přijetí. Takové větve jsou různě dlouhé, takže ne všechny musí odhalovat skutečnou příčinu zamítnutí.

Příklad 5.1. Nechť $L = \{a^{2^n} | n \geq 1\}$ je jazyk popsaný stavovou gramatikou

$$G = (\{S, A, X, a\}, \{s, p, q, m, f\}, \{a\}, R, S, s)),$$

kde R obsahuje nsáledující pravidla:

$$
\begin{array}{ll}
(s,S) \to (p,AAX) & (p,A) \to (q,S) \\
(p,A) \to (f,a) & (q,A) \to (q,S) \\
(f,A) \to (f,a) & (q,X) \to (m,X) \\
(f,X) \to (f,\varepsilon) & (m,S) \to (m,AA) \\
 & (m,X) \to (p,X)
\end{array}
$$

Při analýze, zda řetězec *aaa* patří do jazyka L bude zjištěno, že v průběhu analýzy jedné větve budou přijaty první 2 symboly *a*, avšak jeden přebývá. Skutečnou chybou ale může být, že jeden symbol *a* naopak schází. V tomto případě je možné vypsat, že chyba nastala u třetího symbolu nebo v důsledku chybějícího 4. symbolu. V tomto konkrétním příkladě jde o vypsání všech chyb analýzy, nicméně rozšíříme–li zadání o analýzu řetězce *aaaaaaa*, lze předpokládat, že chyba je pouze jedna a to v chybějícím osmém symbolu *a*. Při analýze však budou zjištěny následující chyby

1. nadbytečný řetězec *aaaaa* na pozici 3
2. nadbytečný řetězec *aaa* na pozici 5
3. chybějící symbol *a* na pozici 8

V tomto konkrétním případě rozhodně nemohlo dojít k chybě č. 1, neboť existuje delší posloupnost symbolů, jenž byla v jiné větvi přijata, a zároveň skončila v koncovém stavu. Nakonec jsem zvolil řešení, kdy za skutečnou chybu budou považovány jen ty, které přijaly největší počet vstupních symbolů. Důvodem je hlavně to, že zatímco při znalosti kontextu vím, že chyba nastala na pozici 5 nebo 8, tak analyzátor může nalézt další chyby v tomto rozmezí, čímž se chyba na pozici 5 „ztratí v davu“. Pokud bych chtěl vypsat i takovou chybu, pravděpodobně by to vedlo i k vypsání chyby na pozici 3, o které vím, že se o skutečnou chybu nejedná.

5.7 Vlastnosti metody

Tato metoda je navržena tak, aby bylo možné analyzovat libovolnou stavovou gramatiku a to i takovou, jenž obsahuje vymazávací pravidla. Návrh vychází z metody backtracking určené pro vyhledávání ve stavovém prostoru. Toto prohledává je, v případě validního vstupu, úplné, takže i metoda syntaktické analýzy takového vstupu je úplná. Aby nedošlo k zacyklení, zejména při nevalidním vstupu, obsahuje tato metoda i detekci rekurze. Aktivací režimu detekce rekurze však může nastat situace, že analyzátor může zamítnout i korektní vstup. Dokud tedy nebude nalezeno ekvivalentní nahrazení veškeré rekurze za nerekurzivní řešení, pak aktivní detekce rekurze způsobí omezení generativní síly této analýzy pravděpodobně do oblasti pod kontextové gramatiky.

Kapitola 6

Implementace

Navržený syntaktický analýzátor jsem implementoval formou konzolové aplikace ve skriptovacím jazyce Python verze 2.7. Tento jazyk jsem si vybral především pro jeho možnosti práce se slovníky a jeho snadnou rozšiřitelnost. Pro implementaci jsem zvolil objektový návrh, jenž mi umožnil abstrahovat řešený problém na úroveň, jenž se velmi blíží matematickému modelu popsanému v předchozích kapitolách.

6.1 Struktura aplikace

Celá aplikace se skládá ze 3 logických částí, tzv. „modulů“. Prvním z nich je modul `structures`, jenž obsahuje definice *tříd* pro jednotlivé matematické struktury gramatiky a dále struktury využívané v průběhu samotné analýzy. Jedná se především o třídy

- `Set` – abstrakce množiny a operací nad ní
- `Rules` a `Rule` reprezentují po řadě množinu pravidel resp. samotná pravidla
- `Stack`, jenž představuje *hluboký zásobník*
- `Config` – datová struktura obsahující aktuální konfiguraci a navíc další informace, jenž jsou unikátní v každém kroce analýzy dané větve a nakonec
- `Manager`, což je správce prostředků pro komunikaci mezi jednotlivými větvemi analýzy (tzv. „procesy“) a řízením analýzy.

Druhý modul (`sgparser`) obsahuje pouze 2 třídy, jejichž úkolem je syntaktická analýza v rámci jedné větve. První z nich, nazvaná `SGParser`, je rozšířením knihovní třídy `multiprocessing.Process` pro práci s více paralelně běžícími procesy. Jedná se o zobecněný analyzátor, jehož funkcionalita vychází z návrhu vyobrazeného na obrázcích 5.2 a 5.5. Druhá třída tohoto modulu, `StrictSGParser`, tento obecný analyzátor upřesňuje. Obsahuje totiž rozšíření, jenž umožňují efektivnější analýzu gramatik, u kterých však předpokládá jejich předchozí předzpracování tak, aby jejich pravidla pro nejlevější derivaci byly jednoznačné. V opačném případě totiž může způsobit nepřijetí jinak validního vstupního řetězce (viz sekce 5.5). Oba společně tvoří tzv. „srdce“ analyzátoru.

Posledním a zároveň klíčovým modulem je `main`. Jedná se o řídící část, jejíž náplní je v prvé řadě ověření vstupních dat a následná inicializace a spuštění analýzy jako takové.

Při spuštění této aplikace je na vstupu vyžadována předchystaná validní stavová gramatika ve formátu XML (přesný formát je popsán v příloze A) a řetězec vstupních symbolů

učených k analýze. Výstup aplikace závisí na parametech spuštění. Není–li to vysloveně zakázáno, pak analyzátor vypíše na výstup řetězec `SUCCESS` v případě úspěšného přijetí vstupní posloupnosti danou gramatikou, resp. je vypsán `ERROR` a seznam nejpravděpodobnějších chyb, pokud k přijetí nedošlo.

Aplikace je koncipována jako mezičlánek překladače tak, že bude zpracovávat kompletní data z lexikálního analyzátoru a v případě úspěšné syntaktické analýzy poskytne posloupnost pravidel vedoucí k přijetí dané vstupní posloupnosti. Tu je možné získat dvojím způsobem. Jednak lze vypsat pravidla v textovém tvaru na stardardní výstup a druhak je lze exportovat ve formátu XML ve stejné podobě, v jaké jsou pravidla strukturována v příloze A.

Kapitola 7

Závěr

V této práci jsem analyzoval vlastnosti stavových gramatik. Zjistil jsem, že existuje velmi mnoho nedeterministických situací, které mohou mít navíc kontextovou závislost. Popsal jsem možné způsoby, jak některé z těchto nejednoznačností z gramatiky odstranit. Dále jsem se věnoval problému rekurze a to zejména v zanořené derivaci, kdy se levá rekurze může rozšířit v rekurzi s pravým kontextem, a na možnosti detekce takové rekurze za běhu analýzy. Získané znalosti jsem využil v návrhu prakticky zaměřeného syntaktického analyzátoru. Tato metoda vychází z vyhledávací metody backtracking, kdy jsou za běhu analýzy dohledávany posloupnosti pravidel vedoucí k přijetí vstupního řetězce. Tento princip jsem upravil tak, aby mohl být prováděn paralelně a mohl tak získat výsledek analýzy rychleji. Zároveň jsem přidal mechanismy na detekci rekurze a popsal postup, jakým je možné upřesnit skutečnou příčinu zamítnutí nevalidního vstupního řetězce. Navrženou metodu jsem posléze implementoval a úspěšně otestoval.

7.1 Otevřené problémy a možnosti dalšího vývoje

V této práci jsem se mimo jiné snažil najít deterministické ekvivalenty některých gramatických konstrukcí, ale nebyl jsem v této oblasti zcela úspěšný. Doposud nevyřešeným problémem totiž zůstalo nalezení spolehlivého způsobu, jak nahradit rekurzi v zanořené derivaci za jednoznačný ekvivalent a vytvořit tak ucelený transformační algoritmus pro determinizaci libovolné stavové gramatiky.

Další vývoj je možné zaměřit na umožnění návratu při výskytu chyby. V případě analýzy bez větvení lze jednoznačně určit chybu a podle ní a aktuálního vstupu odhadnout, kam je možné se vrátit. Naproti tomu u větvené analýzy, jako v případě zde navržené metody, je výskytů chyb více (viz příklad 5.1). Skutečný výskyt se dohledává až po ukončení analýzy všech větví. Pokud by ale existoval mechanismus, který by určil, zda detekovaná chyba v aktuální větvi je opravdovou příčinou zamítnutí a to ještě za běhu dané větve, bylo by možné provést návrat a pokračovat dál v analýze.

Z hlediska samotné implementace vidím rezervu v počtu větvení analýzy. Možnost další redukce počtu větvení spatřuji v předzpracovaní lexikální analýzy a zavedení prediktivní detekce chyb založené na porovnávání počtu terminálů na vstupu vůči těm v derivačním řetězci. Otázkou ovšem je, jaký vliv by tato detekce chyb měla na celkovou výpočetní složitost.

Literatura

[1] Fowler, M.: *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, třetí vydání, ISBN 0-321-19368-7.

[2] Horvat, G.; Meduna, A.: On State Grammars. *Acta Cybernetica*, ročník 1988, č. 8, 1988: s. 237–245, ISSN 0324-721X.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=6156

[3] Jones, M. T.: *Artificial Intelligence: A Systems Approach: A Systems Approach*. Jones & Bartlett Learning, 2008.

[4] Kasai, T.: *An hierarchy between context-free and context-sensitive languages*, ročník 4. Elsevier, 1970, 492–508 s.

[5] Meduna, A.: *Prostředky pro zvýšení generativní síly bezkontextových gramatik*. Dizertační práce, Vysoké učení technické v Brně, Fakulta elektrotechnická, 1987.

[6] Meduna, A.: *Automata and Languages: Theory and Applications [Springer, 2000]*. Springer Verlag, 2005, ISBN 1-85233-074-0.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=6177

[7] Meduna, A.: Deep Pushdown Automata. *Acta Informatica*, ročník 2006, č. 98, 2006: s. 114–124, ISSN 0001-5903.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8000

[8] Meduna, A.: *Elements of Compiler Design*. Taylor and Francis, Taylor & Francis Informa plc, 2008, ISBN 978-1-4200-6323-3.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8538

[9] Meduna, A.; Zemek, P.: *Regulated Grammars and Their Transformations*. Brno University of Technology, 2010, ISBN 978-80-214-4203-0.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=9520

[10] Russell, S. J.; Norvig, P.; Canny, J. F.; aj.: *Artificial intelligence: a modern approach*, ročník 74. Prentice hall Englewood Cliffs, 1995.

[11] Zbořil, F.; Zbořil ml., F.: *Základy umělé inteligence IZU*. Studijní opora, Vysoké učení technické v Brně, Fakulta informačních technologií, třetí vydání, 2006.
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/IZU-IT/texts/oporaIZU-ESF-4.pdf

[12] Zemek, P.: *Kanonické derivace programovaných gramatik*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2008.

Příloha A

Formát XML souboru s gramatikou

```
<?xml version="1.0" encoding="utf-8"?>
<grammar>
	<!-- blok pro stavy gramatiky -->
	<states>
		<!-- ukazka definice konkretniho stavu 'p' -->
		<symbol>p</symbol>
	</states>
	<!-- blok pro abecedu terminalu -->
	<terminals>
		<!-- ukazka definice konkretniho terminalu 'a' -->
		<symbol>a</symbol>
	</terminals>
	<!-- blok pro abecedu neterminalu -->
	<nonterminals>
		<!-- ukazka definice konkretniho neterminalu 'A' -->
		<symbol>A</symbol>
	</nonterminals>
	<!-- pocatecni stav analyzy -->
	<starting_state>s</starting_state>
	<!-- pocatecni neterminalni symbol -->
	<starting_symbol>S</starting_symbol>
	<!-- pravidla gramatiky-->
	<rules>
		<!-- ukazka definice pro pravidlo (s, S) -> (p, AB) -->
		<rule from_state="s" from_symbol="S" to_state="p">
			<!-- derivacni retezec definovany
			po jednotlibych symbolech -->
			<symbol>A</symbol>
			<symbol>B</symbol>
		</rule>
	</rules>
</grammar>
```

Příloha B

Příklad syntaktické analýzy struktur, které nejsou bezkontexové

Pro demonstraci fungování navržené metody jsem zvolil 2 jazykové konstrukce, které nejsou bezkontextové. Jedná se o jazyky $L_1 = \{ww|w \in \{a,b\}^+\}$ a $L = \{a^n b^n c^n | n \geq 0\}$. Tyto konstrukce jsem následně propojil do jazyka $L = \{ww|w = a^n b^n c^n, n \geq 1\}$. Gramatika $G(L)$, popisující tento jazyk, je definována následovně

$$G = (\{S, A_1, B_1, A_2, B_2, a, b, c\}, \{s, f, p, q_1, q_2, q_3, \}, \{a, b, c\}, R, S, s)),$$

kde R obsahuje následující pravidla:

$$\begin{aligned}
(f, B_1) &\to (f, bc) & (q_2, A_2) &\to (q_3, aA_2) \\
(q_1, B_1) &\to (q_2, bB_1c) & (s, S) &\to (p, A_1B_1A_2B_2), \\
(f, A_2) &\to (f, a) & (p, A_1) &\to (q_1, aA_1), \\
(q_3, B_2) &\to (p, bB_2c) & (p, A_1) &\to (f, a), \\
(f, B_2) &\to (f, bc)
\end{aligned}$$

Syntaktická analýza řetězce $aabbccaabbcc \in L$ bude navrženou metodou vypadat následovně:

(s, S)	$\Rightarrow$	$(p, A_1B_1A_2B_2)$	$[(s, S) \to (p, A_1B_1A_2B_2)]$
	$\Rightarrow$	$(q_1, aA_1B_1A_2B_2)$	$[(p, A_1) \to (q_1, aA_1)]$
	$\Rightarrow$	$(q_2, aA_1bB_1cA_2B_2)$	$[(q_1, B_1) \to (q_2, bB_1c)]$
	$\Rightarrow$	$(q_3, aA_1bB_1caA_2B_2)$	$[(q_2, A_2) \to (q_3, aA_2)]$
	$\Rightarrow$	$(p, aA_1bB_1caA_2bB_2c)$	$[(q_3, B_2) \to (p, bB_2c)]$
	$\Rightarrow$	$(f, aabB_1caA_2bB_2c)$	$[(p, A_1) \to (f, a)]$
	$\Rightarrow$	$(f, aabbccaA_2bB_2c)$	$[(f, B_1) \to (f, bc)]$
	$\Rightarrow$	$(f, aabbccaabB_2c)$	$[(f, A_2) \to (f, a)]$
	$\Rightarrow$	$(f, aabbccaabbcc)$	$[(f, B_2) \to (f, bc)]$

Tohle ovšem není jediná větev, která v průběhu analýzy vznikne. Podtržené řádky ukazují konfigurace, kde je možné aplikovat více pravidel. Shodou okolností se v obou případech jedná o dvojici pravidel $(p, A_1) \to (q_1, aA_1)$ a $(p, A_1) \to (f, a)$. Při prvním větvení

$$\begin{array}{lcll}(p, A_1B_1A_2B_2) & \Rightarrow & (p, aB_1A_2B_2) & [(p, A_1) \to (f, a)] \\ & \Rightarrow & (f, abA_2B_2) & [(f, B_1) \to (f, bc)]\end{array}$$

bylo použito pravidlo $(p, A_1) \to (q_1, aA_1)$. Posloupnost pravidel aplikovatých v této větvi od okamžiku vzniku je následující

Už po aplikaci druhé pravidla vzniká konflikt, neboť terminálový prefix abderivačního řetězce neodpovídá vstupnímu řetězci, kde je jako druhý symbol očekáván symbol a. Tato větev je tedy slepá.

K druhému větvení došlo po aplikaci 5. pravidla původní větve. Opět došlo k větvení stejných pravidel a tentokrát vedla k přijetí vstupního řetězce aplikace pravidla $(p, A_1) \to (f, a)$. Větev vzniklá aplikací pravidla $(p, A_1) \to (q_1, aA_1)$ vypadala následovně

$$\begin{array}{lcll}(p, aA_1bB_1caA_2bB_2c) & \Rightarrow & (q_1, aaA_1bB_1caA_2bB_2c) & [(p, A_1) \to (q_1, aA_1)] \\ & \Rightarrow & (q_2, aaA_1bbB_1ccA_2B_2) & [(q_1, B_1) \to (q_2, bB_1c)] \\ & \Rightarrow & (q_3, aaA_1bbB_1ccaaA_2B_2) & [(q_2, A_2) \to (q_3, aA_2)] \\ & \Rightarrow & (p, aaA_1bbB_1ccaaA_2bbB_2cc) & [(q_3, B_2) \to (p, bB_2c)] \\ \hline & \Rightarrow & (f, aaabbB_1ccaaA_2bbB_2cc) & [(p, A_1) \to (f, a)]\end{array}$$

Zde analýza končí, neboť byl vygenerován derivační řetězec s terminálovým prefixem aaa, který neodpovídá prvním třem symbolům na vstupu (aab). Během aplikace výše zmíněných pravidel však došlo ještě k jednomu větvení, v pořadí již třetímu. Tato větev byla velice krátká a také skončila nepřijetím a to ze stejného důvodu, jako předchozí větev.

$$(p, aaA_1bbB_1ccaaA_2bbB_2cc) \quad \Rightarrow \quad (q_1, aaaA_1bbB_1ccaaA_2bbB_2cc) \quad [(p, A_1) \to (f, a)]$$

Celkem tedy v této analýze vznikly 4 větve analýzy, z nichž jedna vedla k přejí vstupu a další 3 nikoliv. Posloupnost pravidel vedoucí k přijetí čítá celkem 9 pravidel, dalších 8 bylo aplikováno ve slepých větvích.

Příloha C

Vysvětlivky k symbolům v diagramech