VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

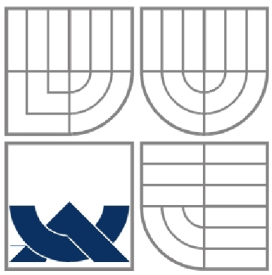
PŘEVODY MEZI CF GRAMATIKAMI A ZÁSOBNÍKOVÝMI AUTOMATY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

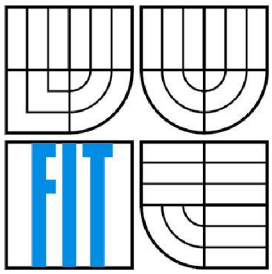
AUTOR PRÁCE
AUTHOR

Bc. BENJAMIN MAKOVSKÝ

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

PŘEVODY MEZI CF GRAMATIKAMI A ZÁSOBNÍKOVÝMI AUTOMATY

CONVERSIONS BETWEEN CF GRAMMARS AND PUSHDOWN AUTOMATA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. BENJAMIN MAKOVSKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

RNDr. TOMÁŠ MASOPUST, Ph.D.

BRNO 2009

Zadání diplomové práce/3896/2008/xmakov01

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2008/2009

Zadání diplomové práce

Řešitel: **Makovský Benjamin, Bc.**
Obor: Informační systémy
Téma: **Převody mezi CF gramatikami a zásobníkovými automaty**
Kategorie: Teorie informatiky

Pokyny:

1. Seznamte se s modely, které se používají v moderní teorii formálních jazyků.
2. Na bázi modelů z předchozího bodu navrhněte webovou aplikaci, která bude převádět bezkontextové gramatiky na zásobníkové automaty a naopak zásobníkové automaty na bezkontextové gramatiky. Vstupní gramatika bude upravena tak, že nebude obsahovat nedosažitelné symboly a bude vlastní, tj. bez nepoužitelných symbolů, bez epsilon-pravidel a necyklická.
3. Aplikaci navrhněte tak, aby ji bylo možno použít jako výukovou pomůcku.
4. Návrh implementujte v jazyce Java/JSP a ověřte jeho správnost.
5. Zhodnoťte dosažené výsledky a diskutujte další možný vývoj projektu.

Literatura:

- Meduna A., Automata and Languages: Theory and Applications Springer, 2000.
- Další dle pokynů vedoucího.

Při obhajobě semestrální části diplomového projektu je požadováno:

- Splnění bodů 1 a 2.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Masopust Tomáš, RNDr., Ph.D.**, UIFS FIT VUT
Datum zadání: 22. září 2008
Datum odevzdání: 26. května 2009

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato práce navrhuje a řeší implementaci transformací bezkontextových gramatik a převody mezi bezkontextovými gramatikami a zásobníkovými automaty. Obsahuje seznámení s modely, které se používají v moderní teorii formálních jazyků. V práci jsou uvedeny všechny potřebné algoritmy pro transformace a vzájemné převody mezi bezkontextovými gramatikami a zásobníkovými automaty. Je představen objektový návrh reprezentující gramatiku a automat v programu. Je popsáno řešení zadávání definice gramatiky a automatu, řešení vykreslování automatu na obrazovku a vytvoření grafického uživatelského rozhraní aplikace. Výsledný program je zpracován jako Java applet, který je umístěn na veřejných internetových stránkách www.convertcfg.php5.cz.

Abstract

This work suggests and solves the implementation of the transformation of context-free grammars and the conversions between context-free grammars and pushdown automata. It makes acquainted with the models used in modern theory of formal languages. In the work are indicated all algorithms necessary for transformations and mutual conversions between context-free grammars and pushdown automata. Proposed is an object representing the grammar and the automaton in the programme. Described is the assigning of definitions of grammar and of the automaton, the solution of drawing the automaton on the screen and the creation of graphical user interface of the application. The resulting programme is developed as Jawa applet which is available on public internet pages www.convertcfg.php5.cz.

Klíčová slova

Abeceda, jazyk, bezkontextová gramatika, neterminální a terminální symbol, přepisovací pravidlo, zásobníkový automat, stav, přechodová funkce.

Keywords

Alphabet, language, context-free grammar, nonterminal and terminal symbol, production rule, pushdown automaton, state, transition relation.

Citace

Makovský Benjamin: Převody mezi CF gramatikami a zásobníkovými automaty, diplomová práce, Brno, FIT VUT v Brně, 2009

Převody mezi CF gramatikami a zásobníkovými automaty

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením RNDr. Tomáše Masopusta, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Benjamin Makovský
4. května 2009

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu mé diplomové práce RNDr. Tomáši Masopustovi, Ph.D. za zájem, připomínky a čas, který věnoval mé práci.

Obsah

1 Úvod

Cílem diplomové práce bylo seznámit se s modely, které se používají v moderní teorii formálních jazyků, především s teorií bezkontextových gramatik, zásobníkových automatů a jejich vzájemné ekvivalence. Vlastní diplomová práce navrhuje a řeší implementaci transformací bezkontextových gramatik a převodu bezkontextových gramatik na zásobníkové automaty a naopak jako Java appletu. Práce svou šíří zasahuje do oblasti teoretické informatiky, programování a počítačové grafiky.

Ve druhé kapitole jsou definovány pojmy a výrazy z teorie formálních jazyků, které jsou důležité pro pochopení problematiky a jsou dále používány v textu. Kapitola směřuje k formálním definicím algoritmů transformace bezkontextové gramatiky na gramatiku vlastní a algoritmů pro vzájemný převod vlastní bezkontextové gramatiky a zásobníkového automatu.

Ve třetí kapitole se zabývám návrhem řešení programu. Je zde představen objektový návrh reprezentující bezkontextovou gramatiku a zásobníkový automat v programu. Jsou popsány důležité metody pro implementaci tříd představujících gramatiku a automat. V této části je také uveden návrh pro grafickou reprezentaci zásobníkového automatu, který je v aplikaci vykreslován. Je představen návrh grafického uživatelského rozhraní celé aplikace.

Z provedené analýzy a návrhu řešení vyplývá implementace v konkrétním programovacím jazyce (Java), která je popsána ve čtvrté kapitole. Je zde uvedeno řešení zadávání definice bezkontextové gramatiky a zásobníkového automatu. Část kapitoly je věnována popisu řešení vykreslování zásobníkového automatu na obrazovku a popisu řešení grafického uživatelského rozhraní.

Závěrečná pátá kapitola obsahuje zhodnocení práce a celkového řešení programu. Je zde uvedeno vyhodnocení testování hotové aplikace a jsou nastíněny další možné vývoje projektu.

V přílohách na konci práce je uveden návod pro používání programu (příloha A), programová dokumentace, která je automaticky vygenerována nástrojem `javadoc.exe` (příloha B). V poslední příloze (C) je popsán obsah přiloženého datového nosiče s nahraným programem.

2 Analýza problému

Pro vlastní implementaci aplikace je nutné seznámit se s modely, které se používají v moderní teorii formálních jazyků, především s teorií bezkontextových gramatik, zásobníkových automatů a jejich vzájemné ekvivalence. V této teoretické části jsem čerpal z literatury [1, 2, 3]. Při studiu transformačních algoritmů pro bezkontextové gramatiky jsem čerpal z literatury [1] a algoritmy pro vzájemný převod bezkontextové gramatiky na zásobníkový automat z literatury [3].

2.1 Základní pojmy

V této podkapitole analýzy problematiky bezkontextových gramatik a zásobníkových automatů jsou definovány pojmy a výrazy z teorie formálních jazyků, které jsou důležité pro pochopení problematiky a jsou dále používány v textu. Jedná se především o pojmy abeceda, jazyk, gramatika a je představena Chomského hierarchie gramatik a jazyků.

2.1.1 Abeceda a jazyk

Abeceda je libovolná konečná neprázdná množina Σ, jejíž prvky nazýváme znaky (písmena, symboly) abecedy. Slovo (též řetězec) nad abecedou Σ je libovolná konečná posloupnost znaků této abecedy. Prázdné posloupnosti znaků odpovídá prázdné slovo, které má nulovou délku a označuje se symbolem ε. Množinu všech slov včetně slova prázdného nad abecedou Σ značíme Σ^*. Symbolem Σ^+ značíme množinu všech slov nad Σ vyjma slova prázdného.

Jazyk nad abecedou Σ je libovolná množina slov nad Σ, jedná se tedy právě o podmnožiny Σ^*. Jazyky mohou být konečné i nekonečné. Konečný jazyk lze reprezentovat výčtem všech jeho slov. Nekonečné jazyky je třeba reprezentovat konečným způsobem, tj. specifikovat množinu všech jeho slov (problém konečné reprezentace). Konečná reprezentace by měla být opět řetězcem symbolů nad jistou abecedou, který budeme vhodně interpretovat tak, aby danému jazyku odpovídala nějaká (ne nutně jediná) konkrétní konečná reprezentace. Takovými reprezentacemi jsou gramatiky a automaty.

Příkladem abecedy je např. množina $\{a, b\}$. Slova nad touto abecedou mohou být *aabb*, *abab*, *bbbaa*, atd. Množina všech slov nad abecedou $\{a, b\}$ je $\{a, b\}^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$. Příkladem nekonečného jazyka nad abecedou $\{a, b\}$ je jazyk $L = \{a^n b^n \mid n \geq 0\}$.

2.1.2 Gramatika

Gramatika reprezentuje konkrétní jazyk. Je to čtveřice (N, T, P, S), kde

- N je abeceda neterminálních symbolů,
- T je abeceda terminálních symbolů taková, že $N \cap T = \varnothing$,

- P je konečná množina pravidel, které obvykle zapisujeme ve tvaru $\alpha \rightarrow \beta$, $\alpha \in (N \cup T)^* N (N \cup T)^*$, $\beta \in (N \cup T)^*$, a čteme „α přepiš na β“,
- S je počáteční neterminální symbol, $S \in N$.

Každá gramatika G určuje binární relaci přímé derivace (přímého odvození) na množině $(N \cup T)^*$, kterou značíme $\Rightarrow_G$. Tato relace je definována $\gamma \Rightarrow_G \delta$ právě když existuje pravidlo $\alpha \rightarrow \beta \in P$ a slova $\eta, \rho \in (N \cup T)^*$ taková, že $\gamma = \eta\alpha\sigma$ a $\delta = \eta\beta\sigma$.

Větné formy gramatiky G jsou prvky množiny $(N \cup T)^*$, které lze odvodit z počátečního neterminálního symbolu. Pro $\alpha \in (N \cup T)^*$ platí, že je větnou formou gramatiky G, pokud $S \Rightarrow_G^* \alpha$. Věta je větná forma, která neobsahuje žádné neterminální symboly. Množina všech vět tvoří jazyk generovaný gramatikou G, označovaný jako $L(G)$: $L(G) = \{w \in T^* \mid S \Rightarrow_G^* w\}$.

2.1.3 Chomského hierarchie gramatik a jazyků

Chomského hierarchie gramatik (a jazyků) rozděluje gramatiky podle jejich popisné síly. Rozlišuje čtyři základní typy gramatik:

- **Typ 0** obsahuje pravidla v nejobecnějším tvaru:
  $\alpha \rightarrow \beta$; $\alpha \in (N \cup T)^* N (N \cup T)^*$, $\beta \in (N \cup T)^*$.
  Někdy se nazývají **gramatikami neomezenými**.
- **Typ 1** obsahuje pravidla tvaru:
  $\alpha A \beta \rightarrow \alpha\gamma\beta$; $A \in N$, $\alpha, \beta \in (N \cup T)^*$, $\gamma \in (N \cup T)^+$ nebo $S \rightarrow \varepsilon$ a S není na žádné pravé straně.
  Nazývají se **kontextovými gramatikami**, poněvadž nonterminální symbol A může být nahrazen řetězcem γ pouze tehdy, je-li jeho levým kontextem řetězec α a pravým kontextem řetězec β.
- **Typ 2** obsahuje pravidla tvaru:
  $A \rightarrow \gamma$; $A \in N$, $\gamma \in (N \cup T)^*$.
  Nazývají se **bezkontextovými gramatikami**, protože nonterminální symbol A může být nahrazen řetězcem γ bez ohledu na kontext, ve kterém je nonterminální symbol A uložen.
- **Typ 3** obsahuje pravidla tvaru:
  $A \rightarrow aB$ nebo $A \rightarrow a$; $A, B \in N$, a $\in T$.
  Nazývají se **regulárními gramatikami**.

Jazyk generovaný gramatikou typu i (i = 0, 1, 2, 3) nazýváme jazykem typu i. Podle názvů gramatik mluvíme o jazycích rekurzivně vyčíslitelných (i = 0), kontextových (i = 1), bezkontextových (i = 2) a regulárních (i = 3).

Nechť L_i, i = 0, 1, 2, 3 značí třídu všech jazyků typu i. Pak platí $L_0 \supset L_1 \supset L_2 \supset L_3$.Každá gramatika typu 1 je zároveň gramatikou typu 0 atd.

Chomského hierarchie gramatik a formálních jazyků je referenčním schématem klasifikace aplikací teorie formálních jazyků a prostředkem vymezujícím popisnou a rozhodovací sílu dané třídy formálních modelů.

2.2 Bezkontextové gramatiky

Bezkontextové gramatiky generují bezkontextové jazyky, které mají praktický význam např. při definici syntaxe programovacích jazyků, formalizaci pojmu syntaktická analýza a návrhu překladu programovacích jazyků.

2.2.1 Definice bezkontextové gramatiky

Bezkontextová gramatika je čtveřice (N, T, P, S) – viz 2.1.2. Přepisovací pravidla z množiny P jsou ve tvaru $A \rightarrow \alpha$, kde $A \in N$, $\alpha \in (N \cup T)^*$. Na levé straně pravidla je libovolný neterminální symbol a na pravé straně je libovolný řetězec složený z neterminálních a terminálních symbolů.

2.2.2 Transformace bezkontextových gramatik

Často bývá výhodné modifikovat danou gramatiku G tak, aby vytvářela takovou strukturu vět z jazyka $L(G)$, která by zajistila splnění některých žádaných vlastností jazyka nebo gramatiky. Jedná se především o převedení libovolné bezkontextové gramatiky na gramatiku vlastní. Pro tento převod je nutné provést transformace odstranění nepoužitelných symbolů, odstranění ε-pravidel a odstranění jednoduchých pravidel. Operace musejí být provedeny v uvedeném pořadí.

V následujících algoritmech představuje označení lhs(p) levou stranu přepisovacího pravidla p a rhs(p) označuje pravou stranu přepisovacího pravidla p.

2.2.3 Nedosažitelné symboly

Abychom mohli eliminovat nepoužitelné symboly, musíme umět odstranit nedosažitelné symboly.

Symbol $X \in (N \cup T)$ je nedosažitelný v gramatice $G = (N, T, P, S)$, jestliže se X nemůže objevit v žádné větné formě. Uvedené algoritmy byly použity v implementaci aplikace.

2.2.3.1 Algoritmus vytvoření množiny dostupných symbolů

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$.

Výstup: Množina W obsahující všechny dostupné symboly gramatiky G.

Metoda: **begin**

$W := \{S\}$;

$OLD := \emptyset$;

repeat

$NEW := W - (OLD \cup T)$;

$OLD := W - T$;

for all $A \in NEW$ **do**

for each $p \in P$ **with** lhs(p) = A **do**

$W := W \cup \{\text{alph (rhs}(p))\}$

until $W - T = OLD$

end.

V algoritmu uvedený symbol alph(rhs(p)) znamená funkci, která rozezná všechny symboly dostupné v pravé straně přepisovacího pravidla.

V prvním kroku je do množiny W (obsahující všechny dostupné symboly v gramatice) přidán startovací symbol S, který je vždy dostupným symbolem. Dále jsou pro všechna pravidla, jejichž symbol na levé straně je neterminálním symbolem obsaženým ve vytvářené množině W (pomocná množina NEW), přidány všechny symboly pravé strany pravidla do množiny W. Jedná se o sjednocení, tedy jsou přidávány jen symboly, které množina ještě neobsahuje. Množina W je vytvářena iterativním způsobem tak dlouho, dokud je do ní přidáván některý prvek (ošetřeno pomocnou množinou OLD).

2.2.3.2 Algoritmus odstranění nedosažitelných symbolů

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$.

Výstup: Bezkontextová gramatika $G_a = (N_a, T_a, P_a, S)$ bez nedosažitelných symbolů, $L(G) = L(G_a)$.

Metoda: **begin**

pomocí algoritmu v kap. 2.2.3.1 sestavíme množinu W všech dostupných symbolů gramatiky;

$N_a := \{A : A \in N \cap W\}$;

$T_a := \{a : a \in T \cap W\}$;

$P_a := \{p : p \in P, \text{lhs}(p) \in N_a, \text{rhs}(p) \in W^*\}$;

sestavíme $G_a = (N_a, T_a, P_a, S)$;

end.

Na začátku je sestavena množina W všech dostupných symbolů gramatiky. Množina neterminálních symbolů výstupní gramatiky je sestavena z neterminálních symbolů vstupní gramatiky, které jsou obsaženy v množině W (průnik množin N a W). Rovněž množina terminálních symbolů je průnikem množiny terminálních symbolů vstupní gramatiky a množiny W. Množina pravidel výstupní gramatiky se sestaví z pravidel vstupní gramatiky, které obsahují na levé straně neterminální symboly obsažené ve výstupní množině neterminálních symbolů N_a a jejichž pravá strana je tvořena pouze symboly množiny W.

2.2.4 Nepoužitelné symboly

Symbol $X \in (N \cup T)$ je nepoužitelný v gramatice $G = (N, T, P, S)$ právě když v G neexistuje derivace tvaru $S \Rightarrow^* wXy \Rightarrow^* wxy$ pro nějaká $w, x, y \in \Sigma^*$.

Odstranění nepoužitelných symbolů probíhá ve dvou krocích. V prvním kroku jsou z gramatiky odstraněny všechny neterminální symboly, které nemohou vygenerovat terminální řetězec. Tento algoritmus je uvedený v kap. 2.2.4.2. V druhém kroku jsou pak odstraněny všechny nedosažitelné symboly podle algoritmu 2.2.3.2. Pořadí použitých algoritmů je podstatné, obrácené pořadí nemusí vést vždy ke gramatice bez nepoužitelných symbolů (ukázáno v kap. 2.2.4.4). Výsledný algoritmus pro odstranění nepoužitelných symbolů je uveden v kap. 2.2.4.3.

2.2.4.1 Algoritmus vytvoření množiny neterminálních symbolů, které lze přepsat na řetězec terminální

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$.

Výstup: Množina V obsahující všechny neterminální symboly, které lze přepsat na terminální řetězec.

Metoda:

begin
  $V := T \cup \{\text{lhs}(p) : p \in P \text{ and } \text{rhs}(p) \in T^*\}$;
  repeat
    $U := V$;
    for all $p \in P$ **such that** $\text{lhs}(p) \notin U$ **do**
      if $\text{rhs}(p) \in U^+$ **then** $V := V \cup \{\text{lhs}(p)\}$
  until $U = V$
end.

Na začátku je množina V inicializována tak, že obsahuje všechny terminální symboly (T) a všechny neterminální symboly obsažené v levých stranách pravidel, jejichž pravé strany jsou tvořeny pouze symboly z terminální množiny (T). Dále jsou procházena všechna pravidla, jejichž levá strana představuje neterminální symbol, který zatím není obsažen v množině V. Pokud je pravá strana takového pravidla tvořena pouze symboly z množiny V, je neterminální symbol představující levou stranu pravidla přidán (sjednocen s množinou V). Množina V je tvořena iterativně v cyklu tak dlouho, dokud je co přidávat. Ukončení cyklu stanovuje rovnost s pomocnou množinou U, do které je na začátku cyklu překopírován obsah množiny V.

2.2.4.2 Algoritmus odstranění neterminálních symbolů, které nemohou vygenerovat terminální řetězec

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$.

Výstup: Bezkontextová gramatika $G_t = (N_t, T, P_t, S)$, $L(G) = L(G_t)$.

Metoda: **begin**

pomocí algoritmu v kap. 2.2.4.1 sestavíme množinu V všech neterminálních symbolů, které lze přepsat na řetězec terminální;

$N_t := \{A : A \in N \cap V\}$;

$P_t := \{p : p \in P, \text{lhs}(p) \in N_t \text{ and } \text{rhs}(p) \in V^*\}$;

sestavíme $G_t = (N_t, T, P_t, S)$;

end.

V prvním kroku je sestavena množina V obsahující neterminální symboly, ze kterých lze derivovat terminální řetězec. Množina neterminálních symbolů výstupní gramatiky je sestavena z neterminálních symbolů vstupní gramatiky, které jsou obsaženy v množině V (průnik množin N a V). Rovněž množina terminálních symbolů výstupní gramatiky je shodná s množinou terminálních symbolů vstupní gramatiky. Množina pravidel výstupní gramatiky se sestaví z pravidel vstupní gramatiky, které obsahují na levé straně neterminální symboly obsažené ve výstupní množině neterminálních symbolů N_t a jejichž pravá strana je tvořena pouze symboly množiny V.

2.2.4.3 Algoritmus odstranění nepoužitelných symbolů

Po uvedení potřebných algoritmů můžeme přistoupit k odstranění nepoužitelných symbolů v bezkontextové gramatice. V implementaci výsledné aplikace jsou použity právě tyto algoritmy.

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$.

Výstup: Bezkontextová gramatika $G_u = (N_u, T_u, P_u, S)$ bez nepoužitelných symbolů, $L(G) = L(G_t)$.

Metoda: **begin**

podle alg. 2.2.4.2 provedeme konverzi $G = (N, T, P, S)$ na $G_t = (N_t, T, P_t, S)$;

podle alg. 2.2.3.2 provedeme konverzi $G_t = (N_t, T, P_t, S)$ na

$G_a = (N_a, T_a, P_a, S)$;

$N_u := N_a$;

$T_u := T_a$;

$P_u := P_a$;

sestavíme $G_u = (N_u, T_u, P_u, S)$;

end.

Při sestavování bezkontextové gramatiky bez nepoužitelných symbolů se nejprve odstraní neterminální symboly, které nemohou vygenerovat terminální řetězec a následně se odstraní nedosažitelné symboly.

Aplikace algoritmu 2.2.3.2 pro odstranění nedosažitelných symbolů musí být provedena až po nalezení gramatiky G_t, opačný postup nemusí vždy vést ke gramatice bez nepoužitelných symbolů (viz následující kapitola).

2.2.4.4 Ukázka použití špatného pořadí algoritmů při odstraňování nepoužitelných symbolů

Uvažujme gramatiku $G = (\{S, A, B\}, \{a, b\}, P, S)$, kde P obsahuje pravidla $S \rightarrow a \mid A$, $A \rightarrow AB$, $B \rightarrow b$.

Aplikujeme-li na G algoritmus 2.2.4.2, získáme $G_t = (\{S, B\}, \{a, b\}, \{S \rightarrow a, B \rightarrow b\}, S)$. Po aplikování alg. 2.2.3.2 obdržíme výslednou ekvivalentní gramatiku $G_u = (\{S\}, \{a\}, \{S \rightarrow a\}, S)$.

Kdybychom jako první aplikovali algoritmus 2.2.3.2, pak zjistíme, že všechny symboly v G jsou dosažitelné a algoritmus 2.2.3.2 tedy gramatiku G nezmění. Po aplikování algoritmu 2.2.4.2 získáme $V = \{S, B\}$, takže výsledná gramatika bude G_t a ne G_u.

2.2.5 ε -pravidla

Další důležitou transformací je odstranění pravidel tvaru $A \rightarrow \varepsilon$ (ε je prázdný řetězec) z gramatiky, kterým říkáme ε-pravidla. Jestliže má jazyk generovaný gramatikou obsahovat také prázdný řetězec, pak není možné, aby gramatika neobsahovala žádné ε-pravidlo. Následující definice zohledňuje tuto skutečnost.

Gramatika $G = (N, T, P, S)$ neobsahuje ε-pravidla, jestliže:

1. P neobsahuje žádné pravidlo tvaru $A \rightarrow \varepsilon$ nebo
2. v P existuje právě jedno pravidlo tvaru $S \rightarrow \varepsilon$ a S se nevyskytuje na pravé straně žádného pravidla z P.

Ke každé bezkontextové gramatice existuje ekvivalentní bezkontextová gramatika bez ε-pravidel, která generuje stejný jazyk. Této skutečnosti využívá algoritmus pro odstranění ε-pravidel uvedený v kap. 2.2.5.2. Tento algoritmus využívá množinu neterminálů, které vedou na derivaci $A \Rightarrow^* \varepsilon$ (kap. 2.2.5.1).

2.2.5.1 Algoritmus vytvoření množiny ε-neterminálů

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$.

Výstup: Množina N_ε obsahující všechny ε-neterminální symboly, $N_\varepsilon = \{A : A \in N \mid A \Rightarrow^* \varepsilon\}$.

Metoda:

```
begin
    N_ε := {lhs(p) : p ∈ P and rhs(p) = ε};
    repeat
        W := N_ε;
        for all p ∈ P with lhs(p) ∈ N – W do
            if rhs(p) ∈ W* then N_ε := N_ε ∪ {lhs(p)};
    until W = N_ε
end.
```

Na počátku se do množiny N_ε vloží všechny neterminální symboly, které tvoří levou stranu pravidel, jejichž pravé strany tvoří pouze prázdný řetězec (ε). V cyklu se do množiny N_ε přidávají levé strany pravidel, jejichž všechny symboly pravé strany jsou již obsaženy v N_ε. Stačí procházet jen pravidla, která mají na levé straně neterminální symbol, který není obsažen v N_ε. Množina N_ε se doplňuje iterativním způsobem tak dlouho, dokud je do ní nějaký symbol přidáván.

2.2.5.2 Algoritmus odstranění ε-pravidel

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$.

Výstup: Bezkontextová gramatika $G_\varepsilon = (N_\varepsilon, T, P_\varepsilon, S)$ bez ε-pravidel), $L(G) = L(G_\varepsilon)$.

Metoda: **begin**

pomocí algoritmu v kap. 2.2.5.1 sestavíme množinu N_ε všech ε-neterminálů;

for all $A \to X_1 \ldots X_n \in P$ **do**

Přidat do P_ε všechna pravidla tvaru $A \to \alpha_1 \ldots \alpha_n \in P$ splňující:

1. pokud $X_i \notin N_\varepsilon$, pak $\alpha_i = X_i$;
2. pokud $X_i \in N_\varepsilon$, pak $\alpha_i = X_i$ nebo $\alpha_i = \varepsilon$;
3. ne všechna α_i jsou ε; (* nepřidávat pravidla typu $A \to \varepsilon$ *);

if $S \in N_\varepsilon$

then přidat do P_ε pravidla $S' \to S \mid \varepsilon$, $S' \notin N \cup \Sigma$; $N_\varepsilon := N \cup \{S'\}$;

else $N_\varepsilon := N$;

sestavíme $G_\varepsilon = (N_\varepsilon, T, P_\varepsilon, S)$;

end.

Na začátku sestavíme množinu N_ε všech ε-neterminálů. Při vytváření množiny pravidel výstupní gramatiky se použijí pravidla původní (vstupní) gramatiky, ve kterých se postupně nahrazují neterminální symboly obsažené v množině N_ε symbolem ε (vynechají se). Nově vzniklá pravidla typu $A \to \varepsilon$ se nepřidávají.

Je-li např. $N_\varepsilon = \{B, C, D, S\}$, pak je pravidlo $B \to bCdS \in P$ nahrazeno pravidly $B \to bCdS$, $B \to bdS$, $B \to bCd$ a $B \to bd$. Při vlastní implementaci bude vhodné použít metodu rekurzivního volání funkce, která bude postupně z daného pravidla odstraňovat neterminální symboly obsažené v množině N_ε.

Pokud se výchozí symbol vstupní gramatiky S nachází v množině N_ε, potom je třeba vytvořit nový výchozí symbol S', přidat jej do množiny neterminálních symbolů výstupní gramatiky a přidat pravidla $S' \to S \mid \varepsilon$ do množiny pravidel výstupní gramatiky.

2.2.6 Jednoduchá pravidla

Pro stanovení vlastní gramatiky je dále potřeba odstranit z gramatiky jednoduchá pravidla. Jednoduchá pravidla v bezkontextové gramatice $G = (N, T, P, S)$ jsou přepisovací pravidla typu $A \rightarrow B$, kde $A, B \in N$.

Na vstupu algoritmu odstraňujícího jednoduchá pravidla (kap. 2.2.6.2) je gramatika bez nepoužitelných symbolů a bez ε-pravidel. Algoritmus sestaví pro každý neterminální symbol gramatiky množinu takových neterminálních symbolů, které vedou na derivaci $A \Rightarrow^* B$, kde $A, B \in N$ (kap. 2.2.6.1).

2.2.6.1 Algoritmus vytvoření množiny neterminálních symbolů, které vedou na derivaci jednoduchého pravidla

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$ bez ε-pravidel a bez nepoužitelných symbolů, $A \in N$.

Výstup: Množina $N_A = \{B : B \in N \mid A \Rightarrow^* B\}$.

Metoda:

begin
  $Old_A := \emptyset$;
  $N_A := \{A\}$;
  repeat
    $New_A := N_A - Old_A$;
    $Old_A := N_A$;
    for each $B \in New_A$ **do**
      if there exists $p \in P$ **such** that $B = \text{lhs}(p)$ and $\text{rhs}(p) \in N$
      then $N_A := N_A \cup \{\text{rhs}(p)\}$;
  until $N_A = Old_A$
end.

Při inicializaci množiny N_A je do ní vložen symbol A, pro který se množina vytváří. V cyklu se procházejí všechna pravidla gramatiky, která na levé straně obsahují prvky množiny N_A (stačí procházet jen nově přidané prvky). V případě, že pravou stranu pravidla tvoří symbol z neterminální množiny N, je tato pravá strana přidána do množiny N_A. Přidání je provedeno sjednocením, které zajistí, že se prvky v množině nebudou opakovat. Množina N_A vzniká iterativně v repeat – until cyklu tak dlouho, dokud je do ní nějaký prvek přidáván.

2.2.6.2 Algoritmus odstranění jednoduchých pravidel

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$ bez ε-pravidel a bez nepoužitelných symbolů.

Výstup: Bezkontextová gramatika $G_U = (N_U, T, P_U, S)$ bez jednoduchých pravidel, $L(G) = L(G_U)$.

Metoda: **begin**

for all $A \in N$ **do**

pomocí algoritmu v kap. 2.2.6.1 sestavíme množinu N_A;

for all $p \in P$ **such that** lhs(p) $\in N_A$ **do**

if rhs(p) $\notin N$ **then** $P_U := P_U \cup \{ A \rightarrow \text{rhs}(p)\}$;

for all $p \in P_U$ **do**

$N_U := N_U \cup \text{lhs}(p)$;

sestavíme $G_U = (N_U, T, P_U, S)$;

end.

Pro každý symbol $A \in N$ z množiny neterminálních symbolů vstupní gramatiky je vytvořena množina N_A a jsou procházena všechna pravidla vstupní gramatiky s tím, že pokud se na levé straně nachází symbol obsažený v N_A a na pravé straně není neterminální symbol z N, pak je do množiny pravidel výstupní gramatiky připojeno pravidlo, ve kterém je na levé straně příslušný symbol A a pravá strana je převzata z tohoto pravidla vstupní gramatiky.

Množina neterminálních symbolů výstupní gramatiky je tvořena všemi levými stranami pravidel výstupní gramatiky. Sjednocení zajistí, že se prvky nebudou opakovat.

2.2.7 Vlastní gramatika

Bezkontextová gramatika $G = (N, T, P, S)$ je necyklická, právě když neexistuje $A \in N$ takový, že $A \Rightarrow^+ A$. Cykly se eliminují odstraněním ε-pravidel a jednoduchých pravidel (algoritmy 2.2.5.2 a 2.2.6.2).

Pokud gramatika G neobsahuje nepoužitelné symboly, ε-pravidla a je necyklická, potom se nazývá gramatikou vlastní. Vlastní bezkontextová gramatika $G = (N, T, P, S)$ splňuje:

1. $N \cup T$ obsahují pouze použitelné symboly,
2. G neobsahuje ε-pravidla,
3. G neobsahuje jednoduchá pravidla.

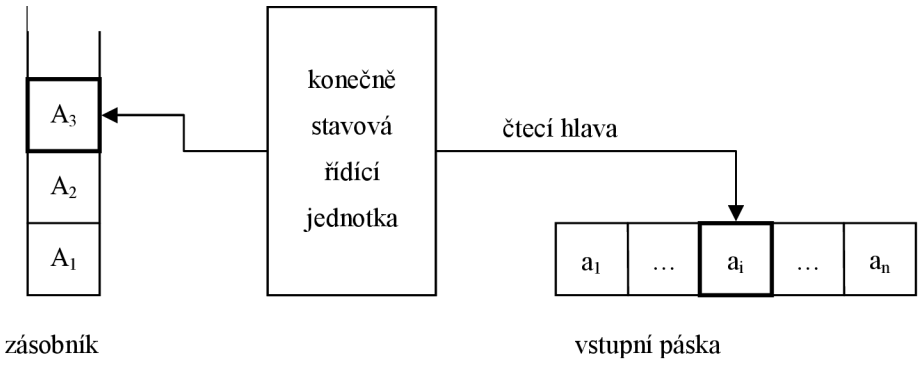
Každou libovolnou bezkontextovou gramatiku lze transformovat užitím výše uvedených algoritmů na gramatiku vlastní.

2.3 Zásobníkové automaty

K bezkontextovým gramatikám existují ekvivalentní zásobníkové automaty, které rozpoznávají jazyky generované těmito gramatikami. Zásobníkový automat si lze představit jako konečný automat,

který obsahuje navíc pomocnou paměť, která pracuje jako zásobník a jejíž velikost není shora omezena (obr. 2.1).

Ze vstupní pásky, na níž je zapsáno slovo nad jistou abecedou, lze pouze číst a čtecí hlava se pohybuje jen vpravo. Na vrchol zásobníku automat ukládá symboly zásobníkové abecedy a takto uložené symboly může následně číst. Automat smí číst pouze z vrcholu zásobníku a přečtený symbol je z vrcholu odstraněn (systém LIFO – Last In First Out). Nelze číst do hloubi zásobníku, aniž by nebyly přečtené symboly odstraněny.



Obr. 2.1 Schéma zásobníkového automatu.

2.3.1 Definice nedeterministického zásobníkového automatu

Nedeterministický zásobníkový automat je sedmice $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kde

- Q je konečná množina stavů,
- Σ je konečná množina vstupní abecedy,
- Γ je konečná množina zásobníkové abecedy,
- δ je zobrazení z množiny $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ do konečné množiny podmnožin množiny $Q \times \Gamma^*$ popisující funkci přechodů,
- $q_0 \in Q$ je počáteční stav,
- $Z_0 \in \Gamma$ je počáteční symbol v zásobníku,
- $F \subseteq Q$ je množina koncových stavů.

2.3.2 Konfigurace zásobníkového automatu

Konfigurace zásobníkového automatu M je trojice $(q, w, \alpha) \in Q \times \Sigma^* \times \Gamma^*$, kde

- q je přítomný stav řídící jednotky,
- w je doposud nepřečtená část vstupního řetězce, je-li $w = \varepsilon$, pak byly všechny symboly ze vstupní pásky přečteny,

- α je obsah zásobníku, je-li $\alpha = \varepsilon$, pak je zásobník prázdný.

2.3.3 Přechod zásobníkového automatu

Přechod zásobníkového automatu reprezentuje binární relace $\vdash$, která je definována na množině konfigurací zásobníkového automatu M. Relace $(q, aw, Z\alpha) \vdash (q', w, \gamma\alpha)$ platí, jestliže $\delta(q, a, Z)$ obsahuje prvek (q', γ) pro nějaké $q \in Q$, $a \in (\Sigma \cup \{\varepsilon\})$, $w \in \Sigma^*$, $Z \in \Gamma$ a $\alpha, \gamma \in \Gamma^*$.

Relaci $\vdash$ interpretujeme tak, že nachází-li se automat ve stavu q a na vrcholu zásobníku je uložen symbol Z, pak po přečtení symbolu $a \neq \varepsilon$ může automat přejít do stavu q' a z vrcholu zásobníku se odstraní symbol Z a vloží se řetězec γ. Je-li $\gamma = \varepsilon$, pak je pouze odstraněn vrchol zásobníku. Čtecí hlava se na pásce posune o jeden symbol vpravo. Je-li $a = \varepsilon$, pozice čtecí hlavy na vstupní pásce se nemění, což znamená, že přechod do nového stavu a nový obsah zásobníku není určován příštím vstupním symbolem.

2.4 Převod bezkontextové gramatiky na zásobníkový automat

Ke každé bezkontextové gramatice $G = (N, \Sigma, P, S)$ lze sestrojit ekvivalentní zásobníkový automat $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ takový, že $L(G) = L(M)$.

2.4.1 Nedeterministický syntaktický analyzátor shora dolů

Zásobníkový automat konstruujeme jako jednostavový, akceptující prázdným zásobníkem tak, aby vytvářel levou derivaci vstupního řetězce v gramatice G. Levé derivace v gramatice G je automat schopen simulovat na zásobníku. Takový automat bude pracovat jako syntaktický analyzátor jazyka generovaného bezkontextovou gramatikou, který pracuje metodou shora dolů.

2.4.1.1 Algoritmus konstrukce zásobníkového automatu (analyzátor shora dolů)

Vstup: Vlastní bezkontextová gramatika $G = (N, \Sigma, P, S)$.

Výstup: Zásobníkový automat akceptující prázdným zásobníkem $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ takový, že $L(M) = L(G)$.

Metoda: $Q := \{q\}$ (* jeden stav q *)

$\Gamma := N \cup \Sigma$

$q_0 := q$

$Z_0 := \varnothing$

δ je definováno takto:

(1) $qA \rightarrow^{\varepsilon} q\alpha \in \delta$, je-li $A \rightarrow \alpha \in P$ (* tj. $\delta(q, \varepsilon, A)$ obsahuje (q, α) *)

(2) $qa \rightarrow^{a} q\varepsilon \in \delta$ pro všechna $a \in \Sigma$ (* tj. $\delta(q, a, a) = \{(q, \varepsilon)\}$ *)

Do množiny stavů výstupního automatu je přidán jeden stav q, který je zároveň počátečním stavem. Vstupní abeceda automatu je shodná s množinou terminálních symbolů gramatiky, zásobníková abeceda je sjednocením množin neterminálních a terminálních symbolů gramatiky. V prvním kroku jsou všechna pravidla gramatiky $A \rightarrow \alpha$ převedena na relace přechodu, kdy se ze zásobníku odebere symbol A a vloží se na zásobník řetězec α, tedy $\delta(q, \varepsilon, A)$ obsahuje (q, α). Ve druhém kroku se pro všechny terminální symboly gramatiky $a \in \Sigma$ vytvoří relace přechodu, kdy je ze vstupu přečten symbol a a ze zásobníku odebrán symbol a s tím, že se na zásobník nic neukládá ($\delta(q, a, a) = \{(q, \varepsilon)\}$).

Ke gramatice G zkonstruujeme jednostavový zásobníkový automat M tak, aby ve svém zásobníku byl schopen simulovat levé derivace v G, přičemž budeme požadovat, aby v každé konfiguraci platilo:

M z konfigurace s obsahem zásobníku α a zbytkem vstupu w

akceptuje prázdným zásobníkem <==> v G lze derivovat $\alpha \Rightarrow^{*} w$.

V G je v jednom kroku odvození nahrazen (nejlevější) neterminál A (naráz, celou) pravou stranou $X_1 \ldots X_n$, kdežto v M bude této situaci odpovídat (1): náhrada neterminálu A na vrcholu zásobníku toutéž pravou stranou následovaná (2): postupným (symbol po symbolu) zpracováním této na vrchol zásobníku přidané pravé strany: nechť se má zpracovat (tj. je na vrcholu zásobníku) X_i. Je-li X_i neterminál, postup (1) opakujeme, je-li (2) X_i terminál, pak zkontrolujeme (tj. ověřuje se korektnost nedeterministické volby v kroku (1)), zda X_i je stejný jako první, dosud nečtený terminál na vstupu. Pokud ano, pak terminál z vrcholu zásobníku odstraníme a čtecí hlava se posune o jeden symbol vpravo.

2.4.2 Syntaktický analyzátor zdola nahoru

Zásobníkový automat konstruujeme se dvěma stavy tak, že vytváří pravou derivaci postupnými redukcemi počínaje terminálním řetězcem umístěným na vstupní pásce a konče výchozím symbolem S. Automat realizuje syntaktickou analýzu zdola nahoru.

2.4.2.1 Algoritmus konstrukce zásobníkového automatu (analyzátor zdola nahoru)

Vstup: Vlastní bezkontextová gramatika $G = (N, \Sigma, P, S)$.

Výstup: Rozšířený zásobníkový automat $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ takový, že $L(M) = L(G)$.

Metoda: $Q := \{q, r\}$

$\Gamma := N \cup \Sigma \cup \{\$\}$

$q_0 := q$

$Z_0 := \$$

$F := \{r\}$

δ je definováno takto:

(1) $\delta(q, a, \varepsilon) = \{(q, a)\}$ pro všechna $a \in \Sigma$

(2) $\delta(q, \varepsilon, \alpha)$ obsahuje (q, A), je-li $A \rightarrow \alpha \in P$

(3) $\delta(q, \varepsilon, S\$) = \{(r, \varepsilon)\}$

Množina stavů automatu je tvořena dvěma stavy q a r (q je počáteční a r koncový stav). Vstupní abeceda automatu je shodná s množinou terminálních symbolů gramatiky, zásobníková abeceda je sjednocením množin neterminálních a terminálních symbolů gramatiky a nového počátečního symbolu na zásobníku $\$$. Přechodová funkce automatu je vytvářena ve třech krocích.

V prvním kroku se pro všechny terminální symboly gramatiky $a \in \Sigma$ vytvoří relace přechodu, kdy je ze vstupu přečten symbol a, ze zásobníku není odebírán žádný symbol a na zásobník se uloží symbol a ($\delta(q, a, \varepsilon) = \{(q, a)\}$). Ve druhém kroku jsou všechna pravidla gramatiky $A \rightarrow \alpha$ převedena na relace přechodu, kdy se ze zásobníku odebere řetězec α a vloží se na zásobník symbol A, tedy $\delta(q, \varepsilon, \alpha)$ obsahuje (q, A). Ve třetím kroku je zásobník vyprázdněn odebráním $S\$$, tj. $\delta(q, \varepsilon, S\$) = \{(r, \varepsilon)\}$.

2.5 Převod zásobníkového automatu na bezkontextovou gramatiku

Ke každému zásobníkovému automatu $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ lze sestrojit bezkontextovou gramatiku $G = (N, \Sigma, P, S)$ takovou, že $L(M) = L(G)$.

Pokud by zásobníkový automat M byl jednostavový, pak by se konstruování gramatiky G provedlo reversním postupem k vytváření zásobníkového automatu v kapitole 2.4.1.1. Proto je třeba sestrojit k danému automatu M ekvivalentní zásobníkový automat M' s jedním stavem takový, že levé odvození v hledané gramatice G má být simulací práce M'. Automat M' simuluje automat M tak, že nedeterministicky hádá, v kterých stavech se M ve svém budoucím výpočtu ocitne, uloží si tato hádání na zásobník s tím, že posléze kontroluje korektnost svého nedeterministického hádání.

2.5.1 Algoritmus převodu zásobníkového automatu na bezkontextovou gramatiku

Vstup: Zásobníkový automat $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$.

Výstup: Bezkontextová gramatika $G = (N, \Sigma, P, S)$ taková, že $L(G) = L(M)$.

Metoda: (1) $N = \{[qZr] \mid q, r \in Q, Z \in \Gamma\} \cup \{S\}$.

(2) Jestliže $\delta(q, a, Z)$ obsahuje $(r, X_1 \ldots X_k)$ $k >= 1$, pak k množině přepisovacích pravidel P přidat všechna pravidla tvaru

$$[qZs_k] \rightarrow a[rX_1s_1]\ [s_1X_2s_2]\ldots\ [s_{k-1}X_ks_k]$$

pro každou posloupnost $s_1, s_2, \ldots, s_k$ stavů z množiny Q.

(3) Jestliže $\delta(q, a, Z)$ obsahuje (r, ε), pak k P přidej pravidlo $[qZr] \rightarrow a$.

(4) Pro každý stav $q \in Q$ přidej k P pravidlo $S \rightarrow [q_0Z_0q]$.

Nejprve se vytvoří taková množina neterminálních symbolů gramatiky, že obsahuje všechny možné kombinace stavů a symbolů zásobníkové gramatiky automatu ($[qZr]$). Je vhodné v názvu neterminálního symbolu uvést názvy stavů a symbol zásobníkové abecedy automatu. Množina terminálních symbolů gramatiky je shodná se vstupní abecedou automatu.

Každé přechodové relaci ze stavu q do stavu r, kdy je přečten ze vstupu symbol a, ze zásobníku je vyzvednut symbol Z a na zásobník je uložen řetězec $X_1 \ldots X_k$, tj. $\delta(q, a, Z)$ obsahuje $(r, X_1 \ldots X_k)$, odpovídají přepisovací pravidla gramatiky tvaru $[qZs_k] \rightarrow a[rX_1s_1]\ [s_1X_2s_2]\ldots\ [s_{k-1}X_ks_k]$ pro každou posloupnost $s_1, s_2, \ldots, s_k$ stavů z množiny Q. Při implementaci je potřeba nalézt všechny možné posloupnosti stavů z množiny stavů a tyto posloupnosti uplatnit při vytváření pravidel.

Každá přechodová relace ze stavu q do stavu r při čtení symbolu a a při odstranění symbolu Z ze zásobníku, tj. $\delta(q, a, Z)$ obsahuje (r, ε), představuje přepisovací pravidlo gramatiky $[qZr] \rightarrow a$.

Pro všechny stavy q automatu se do gramatiky přidají pravidla tvaru $S \rightarrow [q_0Z_0q]$, kde q_0 je počáteční stav automatu a Z_0 je počáteční symbol na zásobníku.

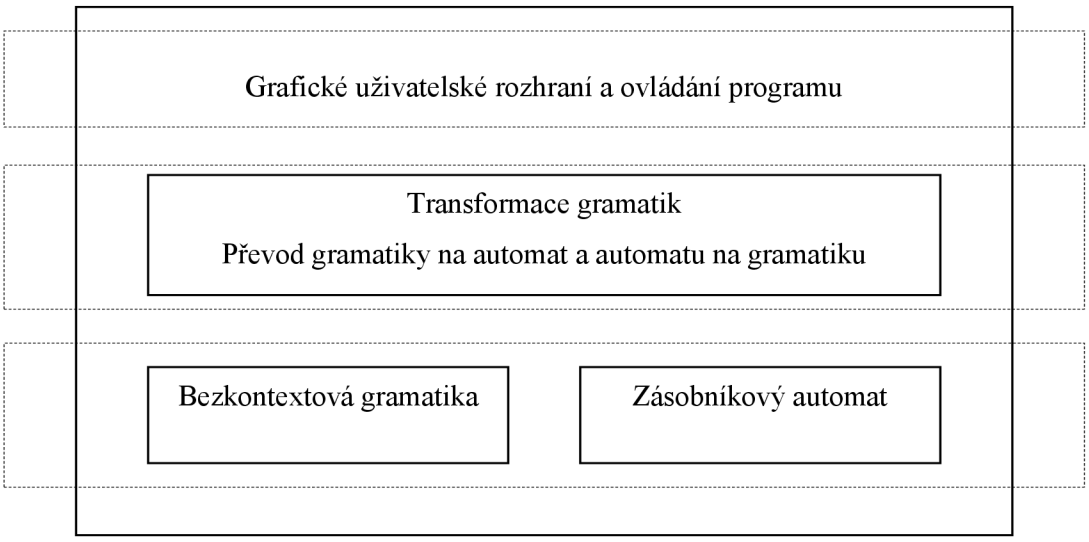
Množina neterminálních symbolů gramatiky N, která se vytvoří v prvním kroku algoritmu, obsahuje některé neterminály, které jsou v gramatice G nedostupné. Abychom je nemuseli dodatečně odstraňovat, začneme v praktickém aplikování algoritmu konstruovat pravidla gramatiky G počínaje S-pravidly a přidáváme pouze ty neterminály, které se objevují na pravých stranách konstruovaných pravidel. S výhodou se při implementaci využije metoda rekurzivního zanořování.

3 Návrh řešení aplikace

Před vlastní implementací aplikace jsem provedl návrh řešení, který je uveden v následujících podkapitolách. Je zde představen vícevrstvý návrh aplikace a objektové řešení instancí bezkontextové gramatiky a zásobníkového automatu.

3.1 Rozdělení návrhu aplikace do více vrstev

Architekturu výsledné aplikace jsem rozdělil do tří vrstev, které na sebe hierarchicky navazují (obr. 3.1). V první vrstvě je řešeno definování, vytvoření a kontrola objektů představujících bezkontextovou gramatiku a zásobníkový automat. V této vrstvě je rovněž obsaženo řešení grafického zobrazení zásobníkového automatu. Druhá vrstva obsahuje řešení vzájemných převodů mezi dvěma gramatikami nebo mezi gramatikou a automatem a naopak. Zde se nachází implementace transformačních algoritmů. Ve třetí vrstvě je řešeno uživatelské rozhraní aplikace, událostní řízení a ovládání programu.



Obr. 3.1 Schéma třívrstvé architektury aplikace.

3.2 Objektový návrh

V aplikaci je navržena třída představující bezkontextovou gramatiku (`CFGrammar`) a třída představující zásobníkový automat (`PDAutomaton`). Obě třídy obsahují atributy odpovídající definicím bezkontextové gramatiky a zásobníkového automatu a slouží pro jejich uložení v paměti.

Přepisovací pravidla bezkontextové gramatiky (podle definice – kap. 2.2.1) jsou objekty třídy `CFGrule`, jejímiž atributy jsou levá a pravá strana pravidla. Mezi třídou představující gramatiku

a třídou přepisovacích pravidel platí vztah kompozice. Gramatika představuje celek, jehož komponenty jsou přepisovací pravidla. Neterminální symboly, terminální symboly a objekty přepisovacích pravidel jsou uloženy v datové struktuře seznam. Startovací symbol je pouze jeden, a proto je uložen v jednoduchém datovém typu.

V případě zásobníkového automatu je navržena třída pro objekty, které představují jeho stavy (`PDAstate`). Každý objekt stavu automatu musí uchovávat kromě svého označení také pozici na vykreslovacím plátně, barvu pro vykreslování a musí znát metody pro vykreslení a zaměření myší. Jednotlivé relace přechodu zásobníkového automatu jsou objekty třídy `PDArule`. V atributech této třídy je uložena levá strana přechodového pravidla a reference na všechny možné její pravé strany, které jsou objekty třídy `PDAruleTrans`. Tento navržený vztah kompozice odpovídá definici funkce přechodu automatu (kap. 2.3.3). Třída reprezentující pravidla umožňuje vykreslování přechodů na kreslící plátno a jejich zaměřování myší.

Objekt zásobníkového automatu sestává z objektů jeho stavů a z objektů jeho přechodových pravidel. Platí zde vztah kompozice. Ostatní atributy podle definice zásobníkového automatu jsou uloženy v datových strukturách seznam nebo v jednoduchých datových typech.

Pro transformace gramatik a pro převod bezkontextové gramatiky na zásobníkový automat a naopak je navržena třída `Conversion`.

3.2.1 Třída `CFGrammar`

Třída `CFGrammar` obsahuje atributy a metody pro objekt představující bezkontextovou gramatiku. Třída bude mít čtyři důležité atributy, které představují čtveřici definice bezkontextové gramatiky (kap. 2.2.1):

- úložiště (seznam) pro uložení neterminálních symbolů,
- úložiště (seznam) pro uložení terminálních symbolů,
- úložiště (seznam) pro uložení objektů přepisovacích pravidel (objekty třídy `CFGrule`),
- proměnná pro uložení startovacího neterminálního symbolu.

Konstruktor třídy vytvoří prázdnou gramatiku, kterou bude moci uživatel naplnit jejími symboly. Druhý přetížený konstruktor vytvoří objekt gramatiky naplněný referenčním zadáním tak, aby bylo možné provést ukázku práce i bez uživatelského zadávání.

Ve třídě existují metody, které zkontrolují uživatelský vstup pro jednotlivé atributy (symboly gramatiky) a propouštějí pouze korektní vstupní údaje. Tyto kontroléry zajistí, aby se neshodovaly názvy neterminálních a terminálních symbolů a aby startovací symbol byl obsažen v množině neterminálních symbolů. Na nekorektní zadání některého údaje bude uživatel upozorněn a tento údaj bude z nadefinování automatu vyřazen.

Pro jednodušší zadávání vstupních hodnot neterminálních a terminálních symbolů je navržen jednoduchý analyzátor, který rozdělí řetězec ve formátu „A,B,C,….,X“ na jednotlivé symboly oddělené symbolem čárka (,).

Třída obsahuje metody pro práci s úložištěm objektů přepisovacích pravidel (objekty třídy `CFGrule`): přidání a odstranění objektu pravidla, nalezení konkrétního pravidla, zjištění existence pravidla.

Dále je zde metoda pro ucelený výpis nadefinované gramatiky, která je využívána pro zobrazení gramatiky v grafickém uživatelském prostředí. Metoda provede transformaci objektové reprezentace gramatiky na textový řetězec.

3.2.2 Třída `CFGrule`

Třída `CFGrule` slouží pro reprezentaci objektu představujícího přepisovací pravidlo gramatiky. Je komponentou třídy `CFGrammar`. Ve třídě jsou navrženy 2 atributy – levá strana a pravá strana přepisovacího pravidla. Oba dva jsou datového typu řetězec (`String`).

Ve třídě existuje metoda, která zadané přepisovací pravidlo tvaru $\alpha \rightarrow \beta$ analyzuje a rozděluje na části α a β (levá a pravá strana pravidla), ve kterých dále analyzuje terminální a neterminální symboly. Jedná se o jednoduchý analyzátor přepisovacích pravidel. Je nutné zkontrolovat, zda zadané přepisovací pravidlo odpovídá pravidlům bezkontextové gramatiky (kap. 2.2.1) a zda všechny analyzované symboly patří do množin terminálních a neterminálních symbolů gramatiky.

Konstruktor třídy vytvoří prázdné přepisovací pravidlo gramatiky. Přetížený konstruktor vytvoří pravidlo gramatiky z předaného řetězce. Při vytváření použije výše popsanou metodu pro analýzu přepisovacího pravidla. Konstruktor se využije převážně při uživatelském zadávání bezkontextové gramatiky. Další přetížený konstruktor nadefinuje přepisovací pravidlo podle předaných parametrů levé a pravé strany pravidla. Tento slouží pro vytváření přepisovacích pravidel v transformačních algoritmech a v algoritmu pro převod automatu na gramatiku.

Dále je zde nadefinována metoda pro zjištění shodnosti dvou přepisovacích pravidel. Shodná pravidla mají shodné levé i pravé strany.

Metoda pro převod přepisovacího pravidla na řetězcovou reprezentaci se využije pro zobrazení gramatiky v grafickém uživatelském prostředí.

3.2.3 Třída `PDAutomaton`

Třída `PDAutomaton` obsahuje atributy a metody objektu představujícího zásobníkový automat.

Třída bude mít sedm důležitých atributů, které představují sedmici definice zásobníkového automatu (kap. 2.3.1):

- úložiště (seznam) pro uložení objektů stavů automatu (objekty třídy `PDAstate`),
- úložiště (seznam) pro uložení vstupní abecedy,

- úložiště (seznam) pro uložení zásobníkové abecedy,
- úložiště (seznam) pro uložení objektů jednotlivých přechodů přechodové funkce automatu (objekty třídy `PDArule`),
- proměnná pro uložení počátečního stavu,
- proměnná pro uložení počátečního symbolu v zásobníku,
- úložiště (seznam) pro uložení koncových stavů.

Konstruktor třídy vytvoří prázdný zásobníkový automat, který bude moci uživatel naplnit jeho definicí. Také zde bude existovat přetížený konstruktor, který vytvoří objekt automatu naplněný referenčním zadáním, aby bylo možné provést ukázku práce i bez uživatelského zadávání.

Ve třídě budou existovat metody, které zkontrolují uživatelský vstup pro jednotlivé atributy (podle definice automatu) a propouštějí pouze korektní vstupní údaje. Tyto kontrolní mechanismy zajistí, aby se neshodovaly názvy stavů s názvy symbolů vstupní a zásobníkové abecedy, aby zadaný počáteční stav byl obsažen v množině stavů a obdobně aby počáteční symbol na zásobníku byl obsažen v množině zásobníkové abecedy. Dále je třeba zajistit, aby množina koncových stavů byla podmnožinou množiny stavů. Na nekorektní zadání některého údaje bude uživatel upozorněn a tento údaj bude z nadefinování automatu vyřazen.

Pro snadnější zadávání vstupních hodnot množiny stavů, vstupní a zásobníkové abecedy a množiny koncových stavů je navržen jednoduchý analyzátor, který rozdělí řetězec ve formátu „A,B,C,….,X“ na jednotlivé symboly oddělené symbolem čárka (,).

Třída obsahuje metody pro práci s úložištěm objektů stavů (objekty třídy `PDAstate`): přidání a odstranění objektu stavu, nalezení konkrétního stavu, zjištění existence stavu. Obdobně pro objekty představující přechody přechodové funkce automatu (objekty třídy `PDArule`) existují metody pro přidání a odstranění přechodu, nalezení konkrétního přechodu a zjištění existence přechodu.

Pro grafické zobrazení automatu slouží metoda, která provede vykreslení objektů stavů a přechodů na kreslící plátno – jsou používány metody pro vykreslení příslušných tříd (`PDAstate` a `PDArule`).

Obdobně pro zjištění, zda je v grafické reprezentaci automatu ukazatelem myši zaměřen stav nebo přechod, je nadefinována metoda, která zjistí zaměření příslušného objektu. Opět jsou používány metody příslušných tříd (`PDAstate` a `PDArule`).

Pro účely textového výpisu nadefinovaného zásobníkového automatu je navržena metoda, která transformuje objektovou reprezentaci automatu na textový řetězec.

3.2.4 Třída `PDAstate`

Třída PDAstate obsahuje atributy a metody pro objekt představující stav automatu. Tento objekt bude mít především implementováno své vykreslení na kreslící plátno a zjištění zaměření ukazatelem myši. Grafickou reprezentací stavu automatu na kreslícím plátně bude kružnice.

Atributy třídy jsou název stavu, umístění středu kružnice na kreslícím plátně a barva, kterou bude kružnice vykreslena.

Konstruktor této třídy vytvoří stav automatu, jeho souřadnicím na kreslícím plátně přiřadí hodnotu (0, 0), která bude změněna při vytváření celého automatu. Druhý – přetížený konstruktor – vytvoří stav automatu se zadanými souřadnicemi na plátně.

Třída obsahuje metodu, která zajišťuje vykreslení kružnice představující stav automatu na kreslící plátno a metodu, která zjišťuje, zda byl stav zaměřen ukazatelem myši.

3.2.5 Třída `PDArule`

Objekt třídy PDArule představuje relaci přechodu zásobníkového automatu (kap. 2.3.3). V atributech je uložena levá strana přechodové relace a seznam objektů, které představují všechny možné pravé strany přechodové relace se shodnou levou stranou (objekty třídy `PDAruleTrans`).

Atributy levé strany přechodu jsou:

- stav, ve kterém se zásobníkový automat nachází,
- symbol, který je přečten ze vstupu,
- symbol, který je uložen na vrcholu zásobníku.

Relace přechodu bude uživatelem zadávána v podobě řetězce znaků - např. $''(q_1, a, Z) = \{ (q_1, \alpha), (q_2, \beta) \}''$. Ve třídě musí existovat metoda, která tento řetězec analyzuje a rozezná jednotlivé symboly levé strany (q_1, a, Z) a jednotlivé řetězce reprezentující pravé strany $(''(q_1, \alpha)'', ''(q_2, \beta)'')$, které jsou předávány při konstrukci objektu pravé strany přechodové relace (objekt třídy `PDAruleTrans`). V případě rozeznaných symbolů je nutné zajistit kontrolu, zda symbol stavu, ve kterém se automat nachází (q_1), existuje v množině stavů automatu. Dále se kontroluje, zda ze vstupu přečtený symbol (a) je symbolem vstupní abecedy automatu a zda symbol z vrcholu zásobníku (Z) je symbolem zásobníkové abecedy automatu.

Ve třídě existují vedle konstruktoru, který vytvoří prázdný objekt přechodové relace, další dva přetížené konstruktory. Jeden vytváří relaci ze zadaného vstupního řetězce a používá metodu pro analýzu vstupního řetězce popsanou v předchozím odstavci. Druhý vytváří přechodovou relaci přímo ze zadaných hodnot. Tento konstruktor slouží převážně pro vytváření přechodových relací v algoritmu pro převod bezkontextové gramatiky na zásobníkový automat.

Pro výpis přechodové funkce automatu se použije metoda, která převádí objektovou reprezentaci přechodové relace na řetězec znaků. Pro pravé strany relací je volána obdobná metoda ze třídy `PDAruleTrans`.

Při vykreslování zásobníkového automatu na kreslící plátno je nezbytná metoda pro grafickou reprezentaci přechodové relace. Objekt relace postupně zavolá vykreslování všech svých pravých částí (objekty třídy `PDAruleTrans`) a provede vykreslení jednotlivých hran grafu. Obdobným

způsobem pracuje také metoda pro zjištění zaměření hrany představující relaci přechodu ukazatelem myši.

3.2.6 Třída `PDAruleTrans`

Pravá polovina relace přechodu zásobníkového automatu je reprezentována objektem třídy `PDAruleTrans`. Mimo uchování v paměti informací o pravé straně přechodu objekt zajišťuje vykreslení hran představujících přechodovou relaci automatu.

V atributech objektu jsou uloženy informace:

- stav, do kterého automat přechází,
- řetězec, který se uloží na vrchol zásobníku.

Dalším důležitým atributem je informace o počtu hran, které začínají a končí ve stejných stavech, případně zda se nejedná o smyčku (shodný počáteční a koncový stav). Tento atribut je používán pro vlastní vykreslení hrany přechodu. Obdobnou funkci má atribut uchovávající barvu pro vykreslování.

Konstruktoru třídy je předán řetězec znaků představujících pravou polovinu relace přechodu zásobníkového automatu – např. $''(q_1, \alpha)''$. Tento řetězec je třeba rozdělit na část představující stav, do kterého automat přechází (q_1) a na část představující řetězec, který se uloží na zásobník (α). Je nutné zkontrolovat, zda existuje v množině stavů automatu zjištěný stav, do kterého automat přechází. V případě řetězce ukládaného na zásobník je potřeba zjistit, zda jsou všechny jednotlivé symboly řetězce obsaženy v zásobníkové abecedě automatu. V případě negativního výsledku kontroly je tento stav uživateli oznámen a požadovaná relace přechodu nebude vytvořena.

Metoda třídy, která zajišťuje řetězcovou podobu pravé strany relace je volána objektem třídy `PDArule`.

V této třídě jsou implementovány metody pro grafické vyjádření relace přechodu zásobníkového automatu. Jedná se o vykreslení hrany mezi dvěma uzly grafu (stavy automatu) nastavenou barvou na kreslící plátno. Součástí vykreslení je popisek hrany a šipka určující orientaci. Rozlišuje se kreslení obloukem mezi dvěma uzly nebo obloukem kolem jednoho uzlu (smyčka). Jsou zde rovněž metody pro zjištění zaměření hrany ukazatelem myši.

3.2.7 Třída `Conversion`

Třída `Conversion` bude pracovat s objekty třídy `CFGrammar` (bezkontextové gramatiky) a s objekty třídy `PDAutomaton` (zásobníkové automaty) a bude zajišťovat transformaci gramatik a převod gramatiky na automat a naopak.

Metody třídy zajišťující transformace gramatik jsou:

- metoda zpracující algoritmus pro odstranění nedostupných symbolů gramatiky (kap. 2.2.3.2), která používá metodu pro vytvoření množiny dostupných symbolů,
- metoda zpracující algoritmus pro odstranění nepoužitelných symbolů (kap. 2.2.4.3),

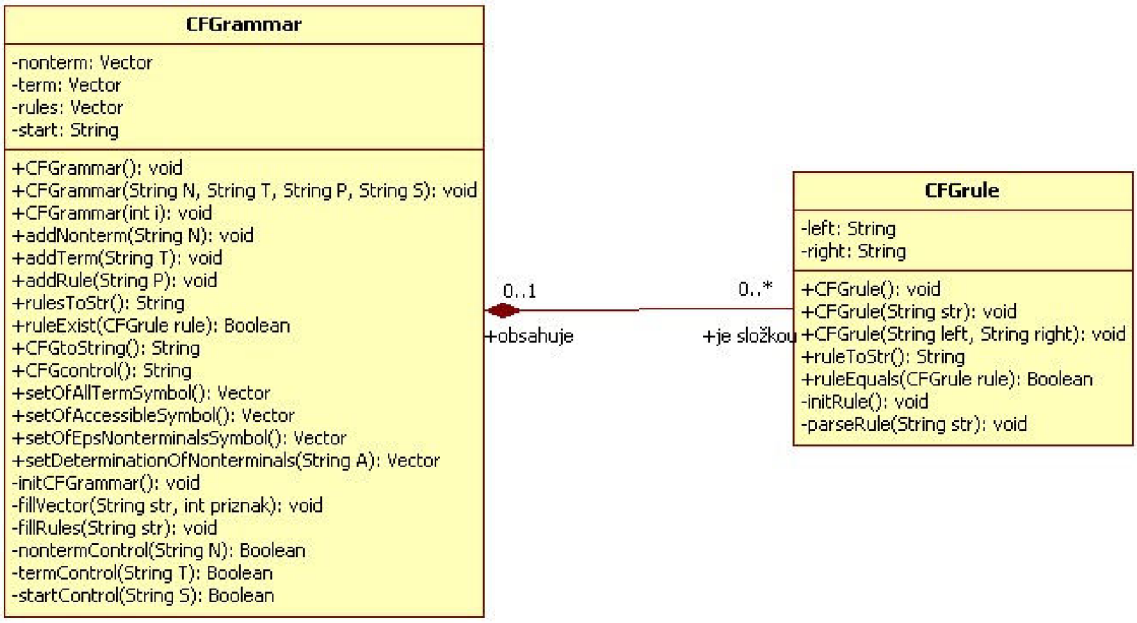
- metoda zpracující algoritmus pro odstranění ε-pravidel (kap. 2.2.5.2), která používá metodu pro vytvoření množiny ε-neterminálů,
- metoda zpracující algoritmus pro odstranění jednoduchých pravidel (kap. 2.2.6.2), která používá metodu pro vytvoření množiny neterminálních symbolů, které vedou na derivaci jednoduchého pravidla,
- metoda zajišťující transformaci zadané bezkontextové gramatiky na gramatiku vlastní, která bude aplikovat výše uvedené metody.

Pro vzájemné převody gramatiky na automat a naopak jsou navrženy dvě metody:

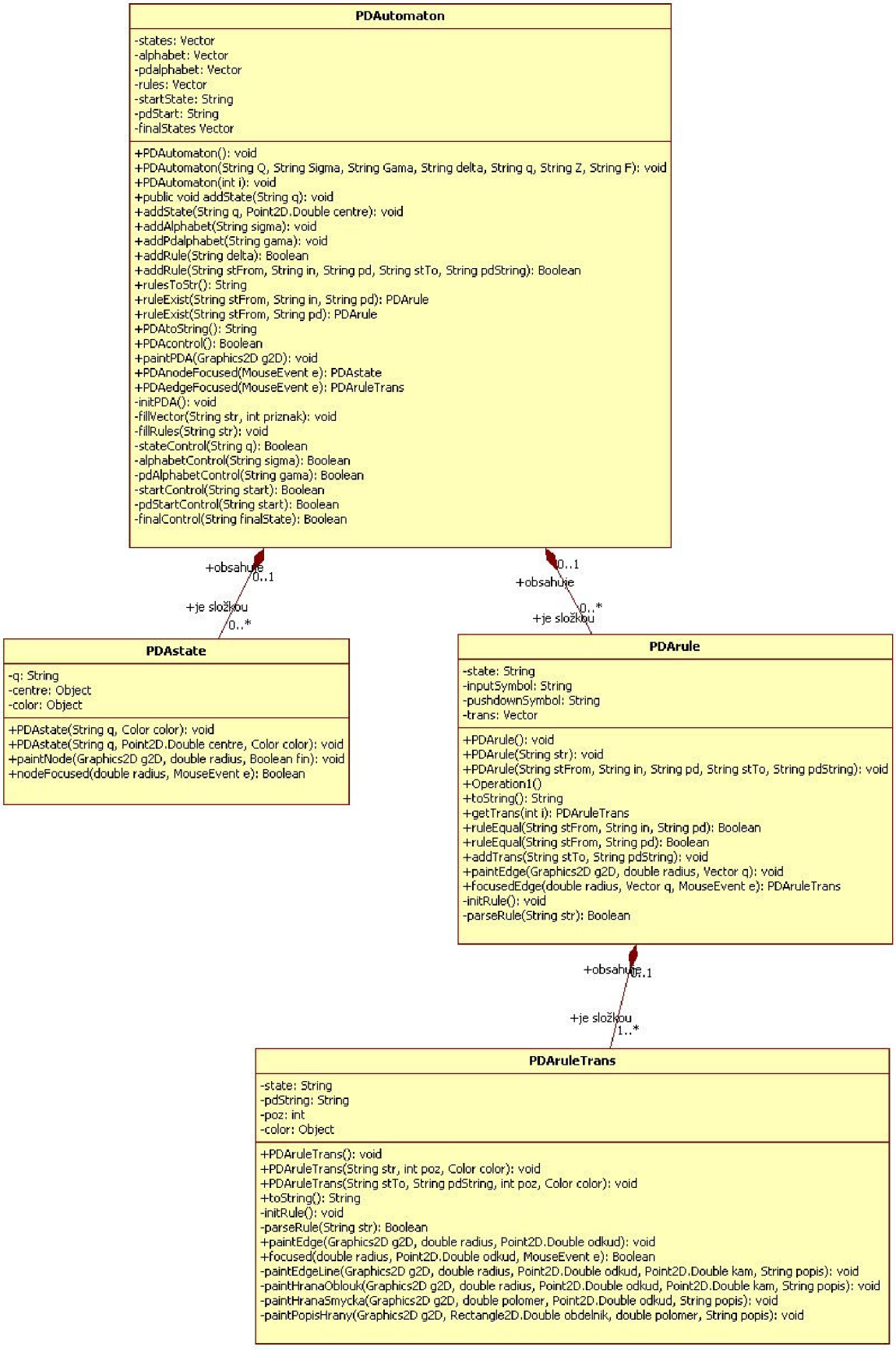
- metoda pro převod bezkontextové gramatiky na zásobníkový automat (kap. 2.4.1.1),
- metoda pro převod zásobníkového automatu na bezkontextovou gramatiku (kap. 2.5.1).

3.3 Navržené diagramy tříd

Diagramy tříd představujících bezkontextovou gramatiku a zásobníkový automat jsou zobrazeny na obr. 3.2 a 3.3. V diagramech jsou pro zjednodušení uvedeny pouze třídy vytvořené speciálně pro reprezentaci gramatiky a automatu.



Obr. 3.2 Diagram tříd pro reprezentaci bezkontextové gramatiky.



Obr. 3.3 Diagram tříd pro reprezentaci zásobníkového automatu.

3.4 Návrh grafického zobrazení zásobníkového automatu

Zásobníkový automat lze vykreslit jako graf. Vrcholy vykresleného grafu představují stavy, ve kterých se automat může nacházet. Hrany grafu představují přechodové relace. Vrcholy se znázorní kružnicemi, vrcholy představující koncové stavy budou vykresleny dvojitou kružnicí. Hrany se vykreslí jako čáry (oblouky), které spojují příslušné dvojice vrcholů (stavů automatu). Orientace hrany bude označena šipkou u koncového vrcholu. Každá hrana bude opatřena popiskem, ze kterého bude možné přečíst parametry přechodové relace.

Při kreslení grafu je dobré dodržovat zásadu, aby hrany byly kresleny pokud možno přímo s co nejmenším počtem průsečíků.

3.5 Návrh grafického uživatelského rozhraní

Grafické uživatelské rozhraní je obrazové rozhraní aplikace. Usnadňuje používání programu tak, že poskytuje konzistentní vnější podobu programu. Mělo by se chovat předvídatelně, aby uživatel věděl, co může po provedení nějaké akce očekávat.

Obrazovka aplikace bude rozdělena na dvě poloviny. V levé polovině bude řešeno zadávání definice a textové zobrazování bezkontextové gramatiky a jejích transformací. V pravé polovině pak zadávání definice a textové i grafické zobrazování zásobníkového automatu. Součástí této části je kreslící plátno pro vykreslení grafické reprezentace automatu.

Zadávání definice gramatiky i automatu bude probíhat zápisem do textových polí. Uživatel bude mít k dispozici tlačítka na přidání nebo odebrání příslušného symbolu definice – neterminální, terminální symboly, přepisovací pravidla, startovací symbol gramatiky, prvky množiny stavů automatu, jeho symboly vstupní a zásobníkové abecedy, jednotlivé relace přechodové funkce, počáteční stav, počáteční symbol na zásobníku a prvky množiny koncových stavů. Pro případ zadávání celých množin bude uživateli k dispozici tlačítko pro provedení rozdělení zadaného řetězce na jednotlivé symboly množiny. Program bude uživatele informovat o všech nekorektních zadáních vstupních údajů.

V aplikaci bude navrženo několik referenčních definicí bezkontextových gramatik a zásobníkových automatů. Uživatel bude moci libovolnou z nich vybrat a dále používat. Důvodem je umožnění demonstrování programu i bez nutnosti zadávání všech parametrů definice.

Kromě textového zadávání definice zásobníkového automatu bude možné automat nadefinovat i v grafickém prostředí pomocí ukazatele myši. V kreslícím plátně lze přidávat stavy a přechody automatu (uzly a hrany grafu) a jejich hodnoty zadávat v dialogovém okně. Vrcholy grafu (stavy automatu) je možné přesouvat po kreslícím plátně pomocí myši. Tak si uživatel může zobrazit

automat podle své představy. Vrcholy i hrany mají své kontextové menu pro snadnou editaci hodnot a odstraňování.

Transformace gramatiky bude realizována výběrem typu transformace a stisknutím tlačítka pro provedení transformace. Ve výstupním okně (určená část obrazovky) bude zobrazena transformovaná gramatika. Algoritmus transformace bude uživateli přehledně představen ve speciálním dialogovém okně.

Převody mezi gramatikou a automatem budou probíhat po stisknutí daného tlačítka. Výstupy převodu budou vždy zobrazeny v druhé polovině obrazovky podle toho, zda půjde o bezkontextovou gramatiku nebo zásobníkový automat. Také zde bude ve speciálním dialogovém okně uživateli předveden algoritmus převodu.

4 Popis řešení

Podle vypracovaného návrhu aplikace vznikla vlastní implementace. V následujících kapitolách jsou uvedeny důležité části implementace jako je řešení analyzátoru přepisovacích pravidel gramatiky a analyzátoru přechodové relace automatu. Je zde popsána implementace vykreslování zásobníkového automatu na obrazovku a implementace uživatelského ovládání programu. V této části jsem čerpal z literatury [4, 5].

4.1 Programovací jazyk

Aplikace je řešena v programovacím jazyku Java, který podporuje dva odlišné typy programů – applety a aplikace. Applet je zvláštní typ programu, který je součástí dokumentu HTML a je možné s ním pracovat v prohlížeči internetových stránek, když je dotyčný dokument načten. Je vhodné, aby applety nebyly příliš rozsáhlé, aby je bylo možné stáhnout ze serveru v krátké době.

Výsledný program je Java appletem. Z bezpečnostních důvodů není appletům dovolen přístup k počítači, na kterém se provádějí. Z tohoto důvodu nemohou číst soubory z disku nebo je zapisovat na disk. Výsledná aplikace tudíž neumožňuje nadefinované objekty gramatiky nebo automatu uchovávat. Výhodou aplikace je její jednoduché zpřístupnění široké veřejnosti přes internetovou síť.

4.2 Datové typy hlavních proměnných

Všechny symboly pro nadefinování bezkontextové gramatiky nebo zásobníkového automatu jsou deklarovány jako znakové řetězce - objekty třídy `String`. Třída `String` je standardní třída, která je součástí Javy a je zvlášť navržena pro vytváření a zpracování řetězců.

Umístění vrcholu grafu (stav automatu) na kreslícím plátně představují reálná čísla typu `double` ve směru osy x a ve směru osy y.

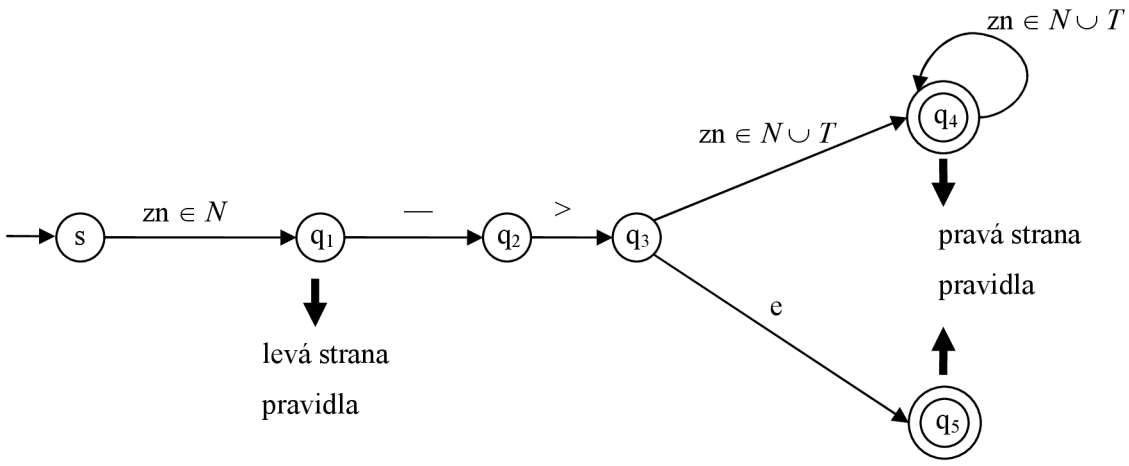
4.2.1 Omezení hodnot proměnných

Z důvodu přehlednosti a snadné čitelnosti zápisu přepisovacích pravidel gramatiky a relací přechodové funkce automatu je vhodné omezit řetězce představující výchozí symboly gramatiky a automatu na délku 1 znak. Pouze v případě demonstrace převodu zásobníkového automatu na bezkontextovou gramatiku je žádoucí, aby vytvořená množina neterminálních symbolů výsledné gramatiky obsahovala víceznakové řetězce (vyplývá to z principu práce algoritmu – kap. 2.5.1). Při vlastní definici gramatiky a automatu, kterou provádí uživatel, je omezena délka vstupního řetězce na jeden znak v případě množiny neterminálních a terminálních symbolů, množiny stavů automatu a množin vstupní a zásobníkové abecedy automatu.

Pro vyjádření prázdného řetězce (kap. 2.1.1) je vyhrazen znak "e", který tedy nebude možné používat v žádné jiné souvislosti.

4.3 Implementace analyzátoru přepisovacích pravidel bezkontextové gramatiky

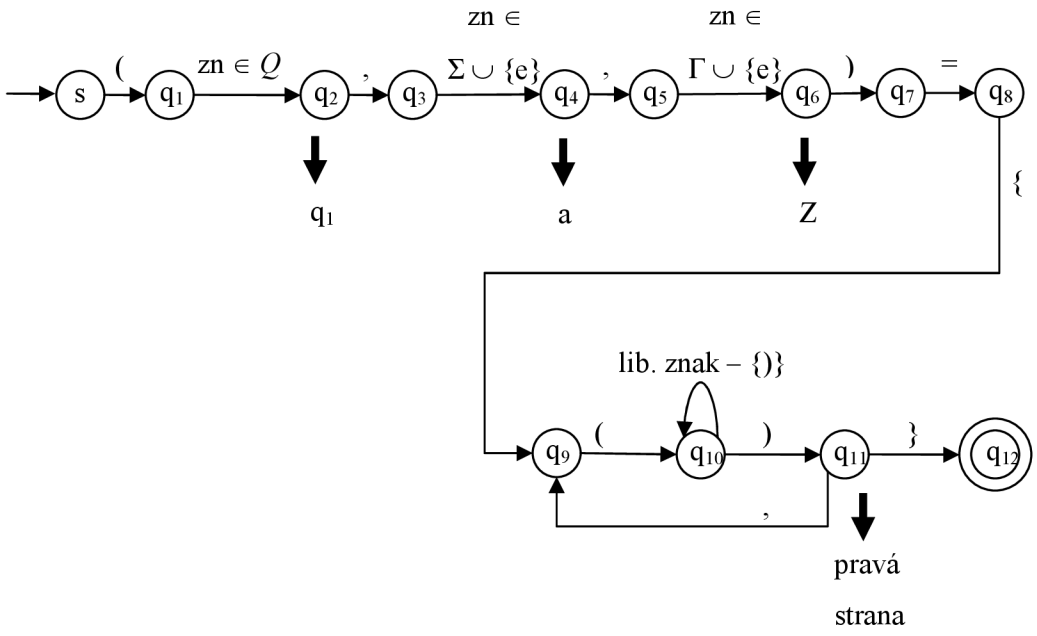
Přepisovací pravidlo je z uživatelského vstupu přečteno jako znakový řetězec (objekt `String`) ve tvaru "A—>α", kde A je znak z množiny neterminálních symbolů (N) a α je řetězec složený ze znaků obsažených v množině neterminálních (N) nebo terminálních symbolů (T) nebo je to znak vyhrazený pro prázdný řetězec - "e". Rozpoznávání jednotlivých jednotek v tomto řetězci je založeno na deterministickém konečném automatu uvedeném na obr. 4.1. V každém stavu je z řetězce přečten 1 znak (v obrázku označen zn). Pokud je ve stavu přečten znak, pro který není definovaný přechod do dalšího stavu, je detekována chyba v zápisu přepisovacího pravidla a přejde se do chybového koncového stavu (pro zjednodušení toto není zakresleno v obrázku). Tučnými šipkami je v obrázku vyznačena situace, kdy je ve stavu přečtena korektní část pravidla, která je zaznamenána (levá a pravá strana přepisovacího pravidla).



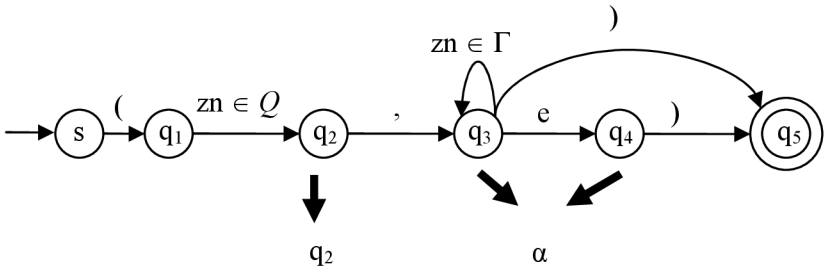
Obr. 4.1 Deterministický konečný automat pro analýzu přepisovacího pravidla.

4.4 Implementace analyzátoru přechodové relace zásobníkového automatu

Relace přechodu je uživatelem zadána v podobě řetězce znaků – např. *"(q1,a,Z) = { (q1,α), (q2,β) }"*, kde *q1* a *q2* jsou znaky z množiny stavů automatu (*Q*), *a* je znak z množiny vstupní abecedy (Σ), *Z* je znak z množiny zásobníkové abecedy (Γ), α a β jsou řetězce znaků, které jsou obsaženy v množině zásobníkové abecedy (Γ). Namístě *a*, *Z*, α a β může být přečten znak označující prázdný řetězec - *"e"*. Deterministický konečný automat pro rozpoznávání levé části přepisovacího pravidla a řetězců pravých částí je zobrazen na obr. 4.2. Pravé části jsou dále analyzovány deterministickým konečným automatem zobrazeným na obr. 4.3. Automaty pracují na stejném principu jako automat v kapitole 4.3. Zaznamenávány jsou přečtené hodnoty *q1* – stav, ve kterém se zásobníkový automat nachází, *a* – symbol přečtený ze vstupu, *Z* – symbol na vrcholu zásobníku, *q2* – stav, do kterého se přechází, α – řetězec, který se ukládá na zásobník. Implementace pracuje na základě definice přechodu (kap. 2.3.3) a návrhu (kap. 3.2.5).



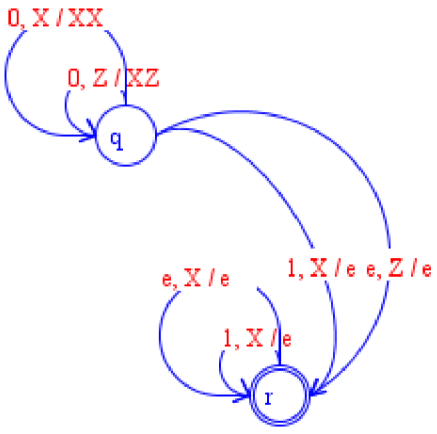
Obr. 4.2 Deterministický konečný automat pro analýzu přechodové relace.



Obr. 4.3 Deterministický konečný automat pro analýzu pravé části přechodové relace.

4.5 Řešení kreslení zásobníkového automatu

Podle návrhu v kap. 3.4 je zásobníkový automat vykreslen jako graf s uzly a hranami. Pro vykreslení grafu je v aplikaci vytvořena instance třídy nazvané `Canvas`, která je odvozena od třídy `JPanel` z balíku `javax.swing` a definuje fyzický panel, do kterého lze umísťovat komponenty. V této třídě je přepsána metoda `paintComponent` tak, aby zobrazovala požadovaná data (stavy a přechody automatu). Metoda aplikuje metodu pro vykreslení automatu ze třídy `PDAutomaton` a tato dále volá metody pro vykreslení stavů a přechodů, které jsou metodami tříd `PDAstate` a `PDArule`. Pro kreslení jednotlivých prvků (čáry, kružnice, oblouky a popisky) jsou použity nástroje balíku `java.awt.geom` pro zobrazování grafiky na obrazovku. Obrázek 4.4 je ukázkou vykresleného zásobníkového automatu.



Obr. 4.4 Ukázka zásobníkového automatu vykresleného řešenou aplikací.

4.5.1 Vykreslení uzlů grafu

Uzly grafu, které představují stavy automatu, jsou vykresleny pomocí kružnic se středem v definované poloze každého vrcholu na kreslícím plátně. Poloměr kružnice je zadán konstantou, která slouží zároveň jako měřítko pro všechny konstantní rozměry v obrázku automatu jako jsou vzdálenosti uzlů, velikosti šipek na hranách, velikost písma v popiscích hran. Změnou této konstanty lze obrázek zvětšovat nebo zmenšovat.

Uzly grafu, které představují koncové stavy automatu, jsou vykresleny dvojitou kružnicí.

Při posunování uzlu tažením myší je přepisována hodnota polohy uzlu na kreslícím plátně a obrázek je překreslen.

4.5.2 Vykreslení hran grafu

Hrany grafu, které představují přechody automatu, jsou vykreslovány třemi způsoby:

1. Úsečkou ležící na přímce spojující dva středové body uzlů grafu.
2. Obloukem spojujícím dvě kružnice uzlů. Použije se v případě, kdy existuje více přechodů mezi dvěma stejnými stavy.
3. Obloukem, který začíná a končí na kružnici představující uzel grafu. Tento způsob slouží k vykreslení smyčky (počáteční a koncový uzel je shodný).

4.5.2.1 První způsob vykreslení hrany - úsečka ležící na přímce spojující dva středové body uzlů grafu

Čára představující hranu grafu je vykreslena jako úsečka spojující dva uzly. Počáteční a koncový bod úsečky leží vždy na obvodu kružnice představující uzel. Tyto body je nutné určit posunutím ze středu kružnice na její obvod ve směru přímky spojující oba středy uzlů. Souřadnice počátečního uzlu označíme $V_1 = (x_1, y_1)$ a koncového uzlu $V_2 = (x_2, y_2)$.

Způsob řešení v programu:

- Počáteční bod úsečky (označíme $A = (x_A, y_A)$) vznikne posunutím středu počátečního uzlu (x_1, y_1) ve směru osy x o hodnotu poloměru kružnice uzlu.
- Souřadnice počátku úsečky: $(x_A, y_A) = (x_1+R, y_1)$.
- Koncový bod úsečky (označíme $B = (x_B, y_B)$) prozatím leží vodorovně s osou x a je posunutý do polohy $(x_A + VZD - 2*R, y_A)$, kde VZD je vzdálenost středů kružnic představující spojované uzly.

  Vzdálenost se vypočítá podle Pythagorovy věty: $VZD = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.
- Souřadnice konce úsečky: $(x_B, y_B) = (x_A + VZD - 2*R, y_A)$.
- Vytvoříme objekt úsečky z bodu A do bodu B: $u = AB = (x_A, y_A, x_B, y_B)$.
- Vytvoříme objekty dvou úseček, které vytvoří šipku na konci úsečky představující hranu.
- První úsečka $u_1 = (x_B - R/2, y_B - R/4, x_B, y_B)$.

- Druhá úsečka $u_2 = (x_B - R/2, y_B + R/4, x_B, y_B)$.
- Vypočítáme úhel ALFA, který svírá přímka spojující středy uzlů s osou x.
- $ALFA = arctan\ (|y_2 - y_1| / |x_2 - x_1|)$.
- Před vykreslením objektů hrany provedeme transformaci otáčení. Uživatelský souřadnicový systém otočíme o úhel *ALFA* kolem bodu představujícího střed kružnice počátečního uzlu. Tato transformace je ekvivalentní se třemi po sobě jdoucími transformačními operacemi – posunutí počátku, otáčení o úhel okolo nové polohy počátku a potom posunutí zpět k obnovení původního počátku.
- Provedeme vykreslení připravených grafických objektů hrany u, u_1, u_2, u_R v transformovaném uživatelském souřadnicovém systému a transformaci vrátíme zpět na původní hodnoty.

4.5.2.2 Druhý způsob vykreslení hrany - obloukem spojujícím dvě kružnice uzlů

Počáteční bod oblouku bude mít hodnotu průsečíku kružnice uzlu a pomyslné přímky vycházející ze středu kružnice pod úhlem 15° vůči ose x. Posunutí ve směru osy x je $\delta x = R * cos(\pi/12)$ a posunutí ve směru osy y je $\delta y = R * sin(\pi/12)$.

Souřadnice počátku oblouku (bod *A*) budou $(x_A, y_A) = (x_1 + \delta x, y_1 + \delta y)$ a souřadnice konce oblouku (bod *B*) budou $(x_B, y_B) = (x_A + VZD - 2* \delta x, y_A)$, kde *VZD* se určí stejně jako v předchozím bodě 4.5.2.1.

Objekt oblouku je definován pomocí obdélníku opsaného elipse, jíž je oblouk součástí. Zadá se počáteční úhel (*0°*), kde oblouk na elipse začíná a úhlová velikost (počet stupňů, které oblouk pokryje - *180°*). Šířka obdélníku (*w*) je rovna hodnotě $VZD - 2* \delta x$, kterou jsme vypočítali v předchozím odstavci. Výška obdélníku (*h*) je závislá na počtu hran, které mají stejný počáteční a koncový uzel. Výška je stanovena násobkem poloměru *R* a pořadím hrany ve skupině hran vycházejících a končících ve stejných uzlech. Horní levý roh opsaného obdélníku má souřadnice $(x_A, y_A - h/2)$, kde *h* je výška obdélníku.

Obdobným způsobem jako v předchozím bodě 4.5.2.1 jsou připraveny objekty šipky, je provedena transformace otáčení o úhel vypočítaný na základě polohy středů spojovaných vrcholů a všechny objekty jsou vykresleny.

4.5.2.3 Vykreslení hrany obloukem (smyčka)

Hrana, která představuje smyčku, je vykreslována obloukem, který je částí kružnice. Oblouk je zadán pomocí obdélníku, do kterého se oblouk vepíše a pomocí počátečního úhlu (zadáno konstantně *0°*), ve kterém oblouk začíná a úhlové velikosti (počet stupňů, které oblouk pokryje, zadáno konstantně *270°*). Horní levý roh opsaného obdélníku má souřadnice konstantně posunuty vůči středovému bodu vrcholu do polohy $(x_1 - 2*k*R, y_1 - 2*k*R)$, kde *R* je opět poloměr kružnice uzlu a k je pořadím hrany ve skupině hran vycházejících a končících v tomto uzlu. Šířka a výška obdélníku je rovna hodnotě *2*R*.

4.5.2.4 Vykreslení popisku hrany

V popisku každé hrany je uveden symbol čteného znaku (a), symbol z vrcholu zásobníku (Z) a řetězec ukládaný na zásobník (α) ve tvaru $a,Z/\alpha$. Popisek je vykreslen do bílého obdélníku, jehož levý horní roh je umístěn ve středovém bodě úsečky nebo oblouku představující hranu. Délka obdélníku závisí na velikosti délky popisku. Popisek se vykresluje zároveň s každou hranou.

Obdélník pro umístění popisku hrany zároveň slouží k určení místa pro zaměření hrany kurzorem myši.

4.6 Řešení grafického uživatelského rozhraní

Jako základní kontejner obsahující jednotlivé komponenty grafického uživatelského rozhraní slouží třída `CFGPDA`, která je podtřídou třídy `javax.swing.JApplet`. Třída `JApplet` navíc implementuje sadu metod, které tvoří rozhraní mezi appletem a webovým prohlížečem.

V aplikaci je zkombinováno více správců rozvržení. Na obrazovce je 6 panelů (instance třídy `JPanel`) se svými komponentami, které seskupují logické celky:

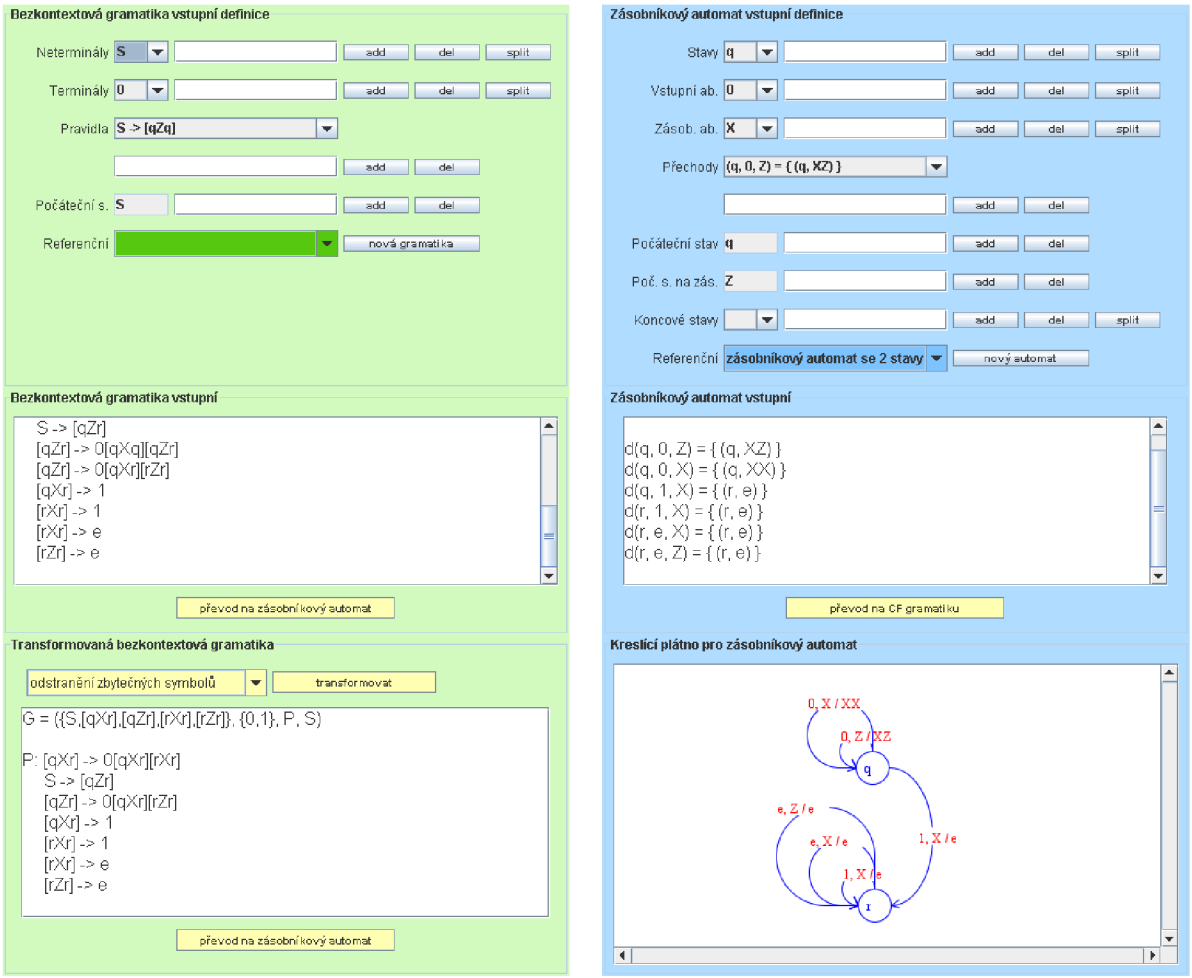
1. Zadávání definice bezkontextové gramatiky, které obsahuje:
   - výběrový seznam, textové pole a tlačítka pro zadání neterminálních symbolů,
   - výběrový seznam, textové pole a tlačítka pro zadání terminálních symbolů,
   - výběrový seznam, textové pole a tlačítka pro zadání přepisovacích pravidel,
   - textové pole a tlačítka pro zadání počátečního symbolu,
   - výběrový seznam pro vybrání referenční bezkontextové gramatiky,
   - tlačítko pro zadání nové definice gramatiky.
2. Textové zobrazení bezkontextové gramatiky, které osahuje:
   - textovou oblast pro výpis nadefinované bezkontextové gramatiky,
   - tlačítko pro spuštění převodu na zásobníkový automat.
3. Transformace bezkontextové gramatiky, které obsahuje:
   - výběrový seznam pro typ transformace,
   - tlačítko pro provedení transformace,
   - textovou oblast pro výpis transformované bezkontextové gramatiky.
4. Zadávání definice zásobníkového automatu, které obsahuje:
   - výběrový seznam, textové pole a tlačítka pro zadání stavů automatu,
   - výběrový seznam, textové pole a tlačítka pro zadání vstupní abecedy,
   - výběrový seznam, textové pole a tlačítka pro zadání zásobníkové abecedy,
   - výběrový seznam, textové pole a tlačítka pro zadání neterminálních symbolů,
   - výběrový seznam, textové pole a tlačítka pro zadání relací přechodové funkce,
   - textové pole a tlačítka pro zadání startovacího stavu,

- textové pole a tlačítka pro zadání počátečního symbolu na zásobníku,
- výběrový seznam, textové pole a tlačítka pro zadání koncových stavů automatu,
- výběrový seznam pro vybrání referenčního zásobníkového automatu,
- tlačítko pro zadání nové definice automatu.

5. Textové zobrazení zásobníkového automatu, které obsahuje:
   - textovou oblast pro výpis nadefinovaného zásobníkového automatu,
   - tlačítko pro spuštění převodu na bezkontextovou gramatiku.
6. Grafické zobrazení zásobníkového automatu, které obsahuje:
   - kreslící plátno pro vykreslení zásobníkového automatu.

V každém panelu jsou komponenty seskupovány do řádků, které jsou rozmístěny pomocí správce rozvržení `BoxLayout`, který dané komponenty rozmisťuje do jednoho sloupce. Zarovnání každého řádku a mezery mezi řádky lze zadat samostatně. Komponenty v řádku jsou rozvrženy pomocí správce `FlowLayout`. Vytvořené komponenty a jejich umístění jsou patrné z obrázku 4.5.

Dále jsou vytvořeny objekty pro „naslouchání“, které budou sledovat a reagovat na události očekávané každou komponentou. Je potřeba přidělit objekty pro naslouchání vhodným komponentám.



Obr. 4.5 Ukázka grafického uživatelského rozhraní aplikace.

4.7 Vlastní implementace

Výsledný applet obsahuje třídu `CFGPDA`, která vytvoří grafické uživatelské rozhraní, třídu `Canvas`, která vytvoří kreslící plátno a vykreslí zásobníkový automat a třídu `HandlerMouse`, která sleduje a obsluhuje události myši.

Třída `CFGPDA` obsahuje metody `init()` a `actionPerformed()`. Metoda `init()` přepisuje metodu `init()` ve třídě `JApplet` a inicializuje grafické uživatelské rozhraní. Bude volána WWW prohlížečem ve chvíli, kdy applet začne svoji činnost. Tato metoda vytvoří instance třídy `CFGrammar a PDAutomaton` - vytvoří prázdnou bezkontextovou gramatiku a prázdný zásobníkový automat. Dále vytvoří objekt pro „naslouchání“ ze třídy `HandlerMouse`, aby sledoval a obsluhoval události generované myší v oblasti obrázku grafu představujícího zásobníkový automat.

Metoda `actionPerformed()` je metodou, kterou volá obsluha události pokaždé, když dojde ke klepnutí myší na některé tlačítko aplikace nebo k výběru některé položky z kontextového menu. Metoda provede akce podle vybrané komponenty.

Objekt třídy `Canvas` je panel, do kterého se provede vykreslení obrázku zásobníkového automatu. Třída je zděděna ze třídy `javax.swing.JPanel`. Vykreslení zajistí metoda `paintComponent`, která volá metodu `paintPDA` objektu zásobníkového automatu. Panel je umístěn do posouvací plochy `scrollPane`, která umožní horizontální a vertikální posouvání v případě, že bude obrázek větší než zobrazené okno.

Vytvořený objekt třídy `HandlerMouse` kontroluje, zda je myší zaměřený uzel nebo hrana grafu představující automat. Umožňuje přesouvání uzlů grafu po kreslícím plátně pomocí držení levého tlačítka myši nebo přidávání hran „klikáním“ levým tlačítkem myši na uzly a otevírá příslušné příruční nabídky (kontextová menu) pro uzel, hranu nebo mimo obrázek grafu.

Ve třídách `CFGrammar` a `CFGrule` jsou implementovány metody pro práci s bezkontextovou gramatikou. Ve třídách `PDAutomaton`, `PDArule` a `PDAruleTrans` se nachází metody objektu zásobníkového automatu a ve třídě `Conversion` jsou metody pro transformaci gramatiky a pro vzájemný převod mezi gramatikou a automatem. Jsou implementovány všechny metody, které jsou popsány v objektovém návrhu v kap. 3.2.

5 Závěr

Důkladně jsem prostudoval literaturu týkající se modelů, které se používají v moderní teorii formálních jazyků, zvláště pak bezkontextových gramatik, zásobníkových automatů a jejich vzájemných převodů. Věnoval jsem se algoritmům transformace bezkontextové gramatiky na gramatiku vlastní a algoritmům převodu bezkontextové gramatiky na zásobníkový automat a naopak.

Z teoretické části (kap. 2 – Analýza problému) vyplývá, že každou bezkontextovou gramatiku lze transformovat na gramatiku vlastní a že platí vztah ekvivalence mezi bezkontextovými gramatikami a zásobníkovými automaty. To znamená, že je možné provádět vzájemné převody mezi gramatikou a automatem a naopak.

Navrhl jsem objektový model řešící definování bezkontextové gramatiky a zásobníkového automatu. Následnou implementací v programovacím jazyku Java vznikl nástroj, ve kterém lze zadat libovolnou bezkontextovou gramatiku nebo zásobníkový automat. Zásobníkový automat je graficky zobrazován vykreslováním na části obrazovky. Na zadané bezkontextové gramatice lze demonstrovat činnost algoritmů pro převod gramatiky na gramatiku bez nepoužitelných symbolů, bez ε-pravidel, bez jednoduchých pravidel a na gramatiku vlastní. Je možné provést převod zadané bezkontextové gramatiky na zásobníkový automat a naopak převod zadaného automatu na gramatiku. Program je napsán jako Java applet a je přístupný na veřejných internetových stránkách www.convertcfg.php5.cz. Zdrojový kód programu včetně programové dokumentace naleznete na přiloženém CD.

Ověření správnosti funkce jednotlivých algoritmů jsem prováděl na příkladech uváděných v literatuře včetně správného řešení, případně jsem správný výsledek ověřoval provedením na papíře. V programu jsem nezjistil žádné nedostatky.

Program lze doplňovat o další ukázky práce s bezkontextovými gramatikami jako jsou např. derivační stromy a fráze větné formy nebo ukázka víceznačnosti gramatiky. Bylo by možné se zabývat algoritmy provádějícími rozklad vět určitého jazyka – syntaktické analyzátory. Algoritmy syntaktické analýzy lze rozdělit podle způsobu, kterým je konstruována derivace věty, na syntaktickou analýzu shora dolů a syntaktickou analýzu zdola nahoru. Transformační algoritmy lze doplnit o transformace do Chomského normální formy nebo do Greibachové normální formy. Rovněž v případě zásobníkových automatů by bylo možné doplnit funkce pro ukázku přijímání řetězců automatem, na kterém by byla demonstrována práce se zásobníkem automatu. Všechna uvedená rozšíření by využily stávajícího objektového modelu pro definování bezkontextové gramatiky a zásobníkového automatu a jeho vykreslování na obrazovku.

Literatura

[1] Meduna, A.: Automata and Languages: Theory and Applications. Springer, 2000.

[2] Černá, I., Křetínský, M., Kučera A.: Automaty a formální jazyky I. Učební text FI MU, 2002, s. 57-104.

[3] Češka, M., Vojnar, T., Smrčka, A.: Teoretická informatika TIN. Studijní opora, 2007.

[4] Horton, I.: Java 5. Praha, Neocortex, spol. s r. o., 2005.

[5] Makovský, B.: Implementace Ford-Fulkersonova algoritmu. Bakalářská práce, Brno, FIT VUT v Brně, 2007.

Seznam příloh

A Návod pro používání programu

A1 Přístup k programu

Aplikace je přístupná na internetových stránkách www.convertcfg.php5.cz. Je zpracována jako Java applet. Stránku lze prohlížet pomocí libovolného prohlížeče s rozšířením pro Java 2. Důležitou podmínkou správného zobrazení stránky je, aby prohlížeč spouštění appletů povoloval.

Pokud se applet na stránce správně nezobrazí, zkontrolujte, zda je spouštění appletů povoleno. Případně je nutné nainstalovat Java Plug-in (J2SE JRE nebo JDK) ze stránek http://java.sun.com/javase/downloads/index.jsp. Nainstalujte poslední verzi, případně 1.4.2 nebo vyšší.

A2 Popis práce s programem

Okno spuštěné aplikace je rozděleno na dvě části. V levé polovině (zelená barva) se nachází definice, zobrazení a obsluha bezkontextové gramatiky. V pravé polovině (modrá barva) je definice, vykreslení a obsluha zásobníkového automatu.

Vlastní definice bezkontextové gramatiky nebo zásobníkového automatu se provádí v horní části příslušného pole. Jsou zde připraveny editační řádky pro všechny definiční symboly, přepisovací pravidla a přechodové relace. Tlačítko `add` symbol nebo pravidlo do definice přidává, tlačítko `del` naopak odebírá symbol nebo pravidlo, které je vybrané v příslušném výběrovém seznamu. Tlačítko `split` provádí přidání všech prvků, které jsou v příslušném editačním řádku zapsány a odděleny čárkami. Je zde také možnost pomocí výběrových seznamů vybrat bezkontextovou gramatiku nebo zásobníkový automat z předdefinovaných možností.

V prostřední části obou polí je prostor pro zobrazení nadefinované bezkontextové gramatiky nebo zásobníkového automatu. Jsou zde umístěna tlačítka pro převod bezkontextové gramatiky na zásobníkový automat a naopak. Výsledek převodu je zobrazen v opačném poli.

V dolní části pole bezkontextové gramatiky se nachází prostor pro transformace gramatiky. Ve výběrovém seznamu lze zvolit typ transformace. Stisknutím tlačítka `transformovat` se provede příslušná transformace vstupní gramatiky a výsledná gramatika se zde zobrazí. Také tuto gramatiku je možné převést na ekvivalentní zásobníkový automat.

Spodní část pole zásobníkového automatu je vyhrazena pro kreslící plátno, ve kterém je zásobníkový automat zobrazován graficky. Pomocí myši lze pohybovat s uzly grafu, které představují stavy automatu. Tímto způsobem je možné dosáhnout požadovaného zobrazení automatu. Kreslící plátno je možné využít také pro vlastní definici zásobníkového automatu pomocí myši. Návod pro kreslení automatu na kreslící plátno je popsán v následující kapitole.

A3 Ovládání kreslení automatu

`Přidat stav:`

Klikněte na volné ploše obrázku pravým tlačítkem. V kontextové nabídce zvolte *nový stav*. V dialogu zadejte označení pro nový stav automatu.

`Přidat přechodovou relaci:`

Levým tlačítkem myši klikněte na počáteční stav a poté klikněte levým tlačítkem myši na koncový stav. V dialogu zadejte hodnoty přechodové relace.

`Zaměřit stav:`

Myší přejeďte dovnitř kružnice uzlu (stavu automatu). Zaměřený uzel změní barvu. Pravým tlačítkem lze vyvolat kontextové menu.

`Zaměřit přechodovou relaci:`

Myší najeďte na popis hrany (přechodová relace automatu). Zaměřená hrana změní barvu.

`Pohyb uzlu (stav automatu):`

Zaměřte uzel, při stisknutém levém tlačítku myši s uzlem pohybujte.

`Odstranit stav:`

Zaměřte uzel (stav automatu), pravým tlačítkem vyvolejte kontextové menu, zvolte *odstranit stav*.

`Nastavit jako koncový stav:`

Zaměřte uzel, pravým tlačítkem vyvolejte kontextové menu, zvolte *koncový stav*.

B Programová dokumentace

Programová dokumentace je automaticky vytvořena nástrojem `javadoc.exe` ze zdrojových souborů programu. Zde je uvedena její část. Celý dokument se nachází na přiloženém CD a lze ho prohlížet v libovolném prohlížeči HTML souborů.

B.1 Package cfgpdaconv

Class Summary	
CFGPDA	Hlavní třída aplikace, vytvoří grafické uživatelské rozhraní.
CFGrammar	Třída CFGrammar, její instance představuje bezkontextovou gramatiku podle definice.
CFGrule	Třída CFGrule, její instance představuje přepisovací pravidlo bezkontextové gramatiky podle definice.
Conversion	Třída Conversion, její instance představuje objekt pro provádění transformací bezkontextových gramatik a pro převod bezkontextové gramatiky na zásobníkový automat a naopak.
PDArule	Třída PDArule, její instance představuje přechodovou relaci zásobníkového automatu podle definice.
PDAruleTrans	Třída PDAruleTrans, její instance představuje pravou stranu přechodové relace zásobníkového automatu podle definice.
PDAstate	Třída PDAstate, její instance představuje stav zásobníkového automatu.
PDAutomaton	Třída PDAutomaton, její instance představuje zásobníkový automat podle definice.

B.2 Class CFGPDA

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Panel
              └java.applet.Applet
                  └javax.swing.JApplet
                      └cfgpdaconv.CFGPDA
```

Hlavní třída aplikace, vytvoří grafické uživatelské rozhraní. Vytvoří bezkontextovou gramatiku - instanci třídy CFGrammar. Vytvoří zásobníkový automat - instanci třídy PDAutomaton. Vytvoří objekt pro převody a transformace - instanci třídy Conversion.

Nested Class Summary

class	**CFGPDA.Canvas** Třída vytvoří kreslící plátno a vykreslí automat.
class	**CFGPDA.HandlerMouse** Třída pro objekt, který sleduje a obsluhuje události generované myší v oblasti obrázku zásobníkového automatu.

Field Summary

static java.lang.String	**EPSILON** Konstanta vyhrazující symbol "e" pro označení prázdného řetězce - epsilon
static java.lang.String	**NEWSTART** Vyhrazený symbol "$" pro nový neterminální symbol

Method Summary

void	**actionPerformed**(java.awt.event.ActionEvent e) Obsluha tlačítek a kontextových menu.
void	**init**() Inicializuje grafické uživatelské prostředí.

B.3 Class CFGPDA.Canvas

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ javax.swing.JComponent
              └ javax.swing.JPanel
                  └ cfgpdaconv.CFGPDA.Canvas
```

Třída vytvoří kreslící plátno a vykreslí automat.

Constructor Summary	
CFGPDA.Canvas() Konstruktor kreslícího plátna.	

Method Summary	
void	**paintComponent**(java.awt.Graphics g) Metoda pro vykreslení zásobníkového automatu na kreslící plátno.

B.4 Class CFGPDA.HandlerMouse

```
java.lang.Object
  └ java.awt.event.MouseAdapter
      └ javax.swing.event.MouseInputAdapter
          └ cfgpdaconv.CFGPDA.HandlerMouse
```

Třída pro objekt, který sleduje a obsluhuje události generované myší v oblasti obrázku zásobníkového automatu. Vytvořený objekt třídy HandlerMysi kontroluje, zda je myší zaměřený stav nebo přechodová relace automatu. Umožňuje přesouvání stavů po kreslícím plátně pomocí držení levého tlačítka myši nebo přidávání přechodů „klikáním“ levým tlačítkem myši na stavy a otevírá příslušné příruční nabídky (kontextová menu) pro stav nebo mimo obrázek automatu.

Constructor Summary	
CFGPDA.HandlerMouse()	

Method Summary	
void	**mouseClicked**(java.awt.event.MouseEvent e) Metoda reagující na kliknutí tlačítka myši.
void	**mouseDragged**(java.awt.event.MouseEvent e) Metoda kontrolující tažení se stlačeným tlačítkem myši.
void	**mouseMoved**(java.awt.event.MouseEvent e) Metoda kontrolující pohyb myši.
void	**mousePressed**(java.awt.event.MouseEvent e) Metoda kontrolující stlačení tlačítka myši.

void	**mouseReleased**(java.awt.event.MouseEvent e) Metoda kontrolující puštění tlačítka myši.

B.5 Class CFGrammar

java.lang.Object
  └ **cfgpdaconv.CFGrammar**

Třída CFGrammar, její instance představuje bezkontextovou gramatiku podle definice. Objekty představující přepisovací pravidla jsou instance třídy CFGrule a jsou uloženy ve vectoru rules. Konstruktor vytvoří bezkontextovou gramatiku. Jsou naimplementovány metody pro ustavení: - množiny neterminálů negenerujících terminální řetězce - množiny dostupných symbolů - množiny eps-neterminálů - množiny neterminálních symbolů, které vedou na derivaci jednoduchého pravidla

Constructor Summary

CFGrammar() Konstruktor, vytvoří prázdnou bezkontextovou gramatiku.
CFGrammar(int i) Přetížený konstruktor, vytvoří referenční bezkontextovou gramatiku.
CFGrammar(java.lang.String N, java.lang.String T, java.lang.String P, java.lang.String S) Přetížený konstruktor, vytvoří nadefinovanou bezkontextovou gramatiku.

Method Summary

void	**addNonterm**(java.lang.String N) Přidá neterminální symbol do množiny neterminálních symbolů.
void	**addRule**(java.lang.String P) Do množiny přepisovacích pravidel přidá pravidlo z parametru P (řetězec).
void	**addTerm**(java.lang.String T) Přidá terminální symbol do množiny terminálních symbolů.
java.lang.Boolean	**CFGcontrol**() Kontrola, zda je nadefinován výchozí symbol gramatiky.
java.lang.String	**CFGtoString**() Převod bezkontextové gramatiky na řetězcovou reprezentaci

void	**delN**(int i) Odstranění neterminálního symbolu na pozici i z množiny neterminálních symbolů.
void	**delRule**(int i) Odstranění přepisovacího pravidla na pozici i z množiny přepisovacích pravidel.
void	**delS**() Odstranění výchozího symbolu.
void	**delT**(int i) Odstranění terminálního symbolu na pozici i z množiny terminálních symbolů.
java.util.Vector	**getNonterm**() Vrací množinu neterminálních symbolů.
java.util.Vector	**getRule**() Vrací množinu přepisovacích pravidel.
CFGrule	**getRule**(int i) Vrací přepisovací pravidlo z množiny přepisovacích pravidel na pozici i.
java.lang.String	**getStart**() Vrací výchozí symbol gramatiky.
java.util.Vector	**getTerm**() Vrací množinu terminálních symbolů.
java.lang.Boolean	**isSetString**(java.lang.String str, java.util.Vector v) Metoda zjistí, zda je vstupní řetězec str složen pouze z prvků množiny v.
java.lang.String	**nontermToStr**() Převede množinu neterminálních symbolů na řetězcovou reprezentaci.
java.lang.Boolean	**ruleExist**(CFGrule rule) Zjišťuje existenci pravidla v parametru rule v množině přepisovacích pravidel.
java.lang.Boolean	**ruleExist**(java.lang.String left) Zjišťuje existenci pravidla, které má levou stranu shodnou s paramtrem left v množině přepisovacích pravidel.

java.lang.String	**rulesToStr**() Převede množinu přepisovacích pravidel na řetězcovou reprezentaci.
java.util.Vector	**setDeterminationOfNonterminals**(java.lang.String A) Metoda iterativním způsobem vytvoří množinu neterminálních symbolů, které vedou na derivaci jednoduchého pravidla pro vstupní neterminální symbol A.
void	**setNonterm**(java.lang.String N) Vytvoří množinu neterminálních symbolů z řetězce předaného v parametru N.
void	**setNonterm**(java.util.Vector v) Vytvoří množinu neterminálních symbolů z množiny předané v paramtru v.
java.util.Vector	**setNt**() Metoda iterativním způsobem vytvoří množinu neterminálů negenerujících terminální řetězce
java.util.Vector	**setOfAccessibleSymbol**() Metoda iterativním způsobem vytvoří množinu dostupných symbolů.
java.util.Vector	**setOfAllTermSymbol**() Metoda iterativním způsobem vytvoří množinu neterminálů negenerujících terminální řetězce
java.util.Vector	**setOfEpsNonterminalsSymbol**() Metoda iterativním způsobem vytvoří množinu eps-neterminálů.
void	**setRules**(java.lang.String P) Vytvoří množinu přepisovacích pravidel z řetězce předaného v parametru P.
void	**setStart**(java.lang.String S) Nastaví výchozí symbol gramatiky.
void	**setTerm**(java.lang.String T) Vytvoří množinu terminálních symbolů z řetězce předaného v parametru T.
void	**setTerm**(java.util.Vector v) Vytvoří množinu terminálních symbolů z množiny předané v paramtru v.
java.lang.String	**termToStr**()

	Převede množinu terminálních symbolů na řetězcovou reprezentaci.
void	**writeCFG**() Výpis definice bezkontextové gramatiky v konzolovém okně.

B.6 Class CFGrule

```
java.lang.Object
  └ cfgpdaconv.CFGrule
```

Třída CFGrule, její instance představuje přepisovací pravidlo bezkontextové gramatiky podle definice. Objekt přepisovacího pravidla ukládá levou a pravou stranu pravidla. Konstruktor vytvoří přepisovací pravidlo. Naimplementována je metoda pro parsování pravidla (levá a pravá strana) z řetězce znaků.

Constructor Summary

CFGrule() Konstruktor, vytvoří prázdé přepisovací pravidlo.
CFGrule(java.lang.String str) Přetížený konstruktor, vytvoří přepisovací pravidlo parsováním z řetězce znaků str.
CFGrule(java.lang.String left, java.lang.String right) Přetížený konstruktor, vytvoří přepisovací pravidlo ze zadané levé strany left a pravé strany right.

Method Summary

java.lang.String	**getLeft**() Vrací levou stranu přepisovacího pravidla.
java.lang.String	**getRight**() Vrací pravou stranu přepisovacího pravidla.
java.lang.Boolean	**ruleEmpty**() Metoda zjišťuje, zda je dané pravidlo nadefinované (existuje levá a pravá strana) nebo je prázdné.
java.lang.Boolean	**ruleEquals**(CFGrule rule) Metoda pro zjištění shodnosti pravidla s předaným pravidlem rule.
java.lang.String	**ruleToStr**() Metoda převede objekt přepisovacího pravidla na jeho řetězcovou

	reprezentaci.
void	**setLeft**(java.lang.String left) Nastaví levou stranu pravidla podle předaného řetězce left.
void	**setRight**(java.lang.String right) Nastaví pravou stranu pravidla podle předaného řetězce right.

B.7 Class PDAutomaton

java.lang.Object
  └ **cfgpdaconv.PDAutomaton**

Třída PDAutomaton, její instance představuje zásobníkový automat podle definice. Objekty představující stavy automatu jsou instance třídy PDAstate. Objekty představující přechodové relace jsou instance třídy PDArule a jsou uloženy ve vectoru rules. Konstruktor vytvoří zásobníkový automat. Jsou naimplementovány metody pro vykreslování automatu na kreslící plátno a pro zaměření částí automatu (stavy a přechody) ukazovátkem myši.

Field Summary

static java.awt.Color	**BASICCOLOR** základní barva prvků
static java.awt.Color	**FOCUSEDCOLOR** barva prvků, které jsou zaměřeny ukazatelem myši
double	**RADIUS** konstanta pro kreslení automatu na plátno, určuje velikost kružnice představující stav
static java.awt.Color	**SELECTEDCOLOR** barva prvku, který je označen

Constructor Summary

PDAutomaton() Konstruktor, vytvoří prázdný zásobníkový automat.
PDAutomaton(int i) Přetížený konstruktor, vytvoří referenční zásobníkový automat.
PDAutomaton(java.lang.String Q, java.lang.String Sigma, java.lang.String Gama,

java.lang.String delta, java.lang.String q, java.lang.String Z, java.lang.String F) Přetížený konstruktor, vytvoří nadefinovaný zásobníkový automat.	

Method Summary

void	**addAlphabet**(java.lang.String sigma) Přidá symbol vstupní abecedy do množiny symbolů vstupní abecedy automatu.
java.lang.Boolean	**addAlphabetEdge**(java.lang.String sigma) Přidá symbol vstupní abecedy do množiny symbolů vstupní abecedy automatu.
void	**addFinalStates**(java.lang.String F) Přidá symbol koncového stavu do množiny koncových stavů automatu.
void	**addPdalphabet**(java.lang.String gama) Přidá symbol zásobníkové abecedy do množiny symbolů zásobníkové abecedy automatu.
boolean	**addPdalphabetEdge**(java.lang.String gama) Přidá symbol zásobníkové abecedy do množiny symbolů zásobníkové abecedy automatu.
boolean	**addPdalphabetString**(java.lang.String pdString) Metoda přidává do množiny symbolů zásobníkové abecedy jednotlivé symboly obsažené ve vstupním řetězci pdString.
void	**addRule**(java.lang.String delta) Do množiny přechodových relací přidá relaci z parametru delta (řetězec).
java.lang.Boolean	**addRule**(java.lang.String stFrom, java.lang.String in, java.lang.String pd, java.lang.String stTo, java.lang.String pdString) Do množiny přechodových relací přidá relaci s předanými parametry.
void	**addState**(java.lang.String q) Přidá stav do množiny stavů automatu.
void	**addState**(java.lang.String q, java.awt.geom.Point2D.Double centre)

	Přidá stav do množiny stavů automatu.
java.lang.String	**alphabetToStr**() Převede množinu symbolů vstupní abecedy automatu na řetězcovou reprezentaci.
void	**delFinalSt**(int i) Odstranění symbolu koncového stavu na pozici i z množiny symbolů koncových stavů automatu.
void	**delGama**(int i) Odstranění symbolu zásobníkové abecedy na pozici i z množiny symbolů zásobníkové abecedy automatu.
void	**delPdStart**() Odstranění výchozího symbolu na zásobníku automatu.
void	**delQ**(int i) Odstranění stavu na pozici i z množiny stavů automatu.
void	**delQ**(PDAstate q) Odstranění stavu zadaného v parametru q z množiny stavů automatu.
void	**delRule**(int i) Odstranění přechodové relace na pozici i z množiny přechodových relací.
void	**delSigma**(int i) Odstranění symbolu vstupní abecedy na pozici i z množiny symbolů vstupní abecedy automatu.
void	**delStartSt**() Odstranění výchozího stavu automatu.
java.lang.String	**finalStatesToStr**() Převede množinu symbolů koncových stavů automatu na řetězcovou reprezentaci.
java.util.Vector	**getAlphabet**() Vrací množinu symbolů vstupní abecedy automatu.
double	**getCanvasHeight**() Vrací výšku kreslícího plátna.
double	**getCanvasWidth**() Vrací šířku kreslícího plátna.

java.util.Vector	**getFinalStates**() Vrací množinu symbolů koncových stavů automatu.
java.util.Vector	**getPDalphabet**() Vrací množinu symbolů zásobníkové abecedy automatu.
java.lang.String	**getPDStart**() Vrací výchozí symbol na zásobníku automatu.
java.util.Vector	**getRule**() Vrací množinu přechodových relací.
PDArule	**getRule**(int i) Vrací přechodovou relaci z množiny přechodových relací na pozici i.
java.lang.String	**getStartState**() Vrací výchozí stav automatu.
PDAstate	**getState**(int i) Vrací stav z množiny stavů automatu na pozici i.
java.util.Vector	**getStates**() Vrací množinu stavů automatu.
java.lang.Boolean	**isSetString**(java.lang.String str, java.util.Vector v) Metoda zjistí, zda je vstupní řetězec str složen pouze z prvků množiny v.
void	**nodeCoordinates**() Určí souřadnice uzlu automatu pro vykreslovaní na plátno.
void	**paintPDA**(java.awt.Graphics2D g2D) Vykreslení automatu na kreslící plátno.
java.lang.Boolean	**PDAcontrol**() Kontrola, zda je nadefinován výchozí stav a výchozí symbol na zásobníku automatu.
PDAruleTrans	**PDAedgeFocused**(java.awt.event.MouseEvent e) Vrací přechod automatu, který je zaměřený myší.
java.lang.String	**pdalphabetToStr**() Převede množinu symbolů zásobníkové abecedy automatu na řetězcovou reprezentaci.
PDAstate	**PDAnodeFocused**(java.awt.event.MouseEvent e) Vrací uzel automatu, který je zaměřený myší.

java.lang.String	**PDAtoString**() Převod zásobníkového automatu na řetězcovou reprezentaci
PDArule	**ruleExist**(java.lang.String stFrom, java.lang.String pd) Zjišťuje existenci přechodové relace podle vstupních parametrů v množině přechodových relací.
PDArule	**ruleExist**(java.lang.String stFrom, java.lang.String in, java.lang.String pd) Zjišťuje existenci přechodové relace podle vstupních parametrů v množině přechodových relací.
java.lang.String	**rulesToStr**() Převede množinu přechodových relací na řetězcovou reprezentaci.
void	**setAlphabet**(java.lang.String sigma) Vytvoří množinu symbolů vstupní abecedy z řetězce předaného v parametru sigma.
void	**setAlphabet**(java.util.Vector v) Vytvoří množinu symbolů vstupní abecedy z množiny předané v paramtru v.
void	**setFinalStates**(java.lang.String F) Vytvoří množinu symbolů koncových stavů z řetězce předaného v parametru F.
void	**setFinalStates**(java.util.Vector v) Vytvoří množinu symbolů koncových stavů z množiny předané v paramtru v.
void	**setPdalphabet**(java.lang.String gama) Vytvoří množinu symbolů zásobníkové abecedy z řetězce předaného v parametru gama.
void	**setPdalphabet**(java.util.Vector v) Vytvoří množinu symbolů zásobníkové abecedy z množiny předané v paramtru v.
void	**setPDStart**(java.lang.String Z) Nastaví výchozí symbol na zásobníku automatu.
void	**setRules**(java.lang.String delta) Vytvoří množinu přechodových relací z řetězce předaného v parametru delta.

void	**setStartState**(java.lang.String q) Nastaví výchozí stav automatu.
void	**setStates**(java.lang.String Q) Vytvoří množinu stavů z řetězce předaného v parametru Q.
java.util.Vector	**stateSeqvences**() Ustavení všech možných různých posloupností symbolů stavů automatu.
java.lang.String	**statesToStr**() Převede množinu symbolů stavů automatu na řetězcovou reprezentaci.
void	**writePDA**() Výpis definice zásobníkového automatu v konzolovém okně.

B.8 Class PDAstate

java.lang.Object
  └ **cfgpdaconv.PDAstate**

Třída PDAstate, její instance představuje stav zásobníkového automatu. Konstruktor vytvoří nový stav zásobníkového automatu. Atributy objektu stavu automatu jsou symol pro označení, pozice pro vykreslování a barva pro vykreslování. Naimplementována je metoda pro vykreslení stavu pomocí kružnice na kreslící plátno a metoda pro zjištění, zda je stav zaměřený myší.

Constructor Summary

PDAstate(java.lang.String q, java.awt.Color color) Konstruktor, vytvoří stav označený symbolem q.
PDAstate(java.lang.String q, java.awt.geom.Point2D.Double centre, java.awt.Color color) Přetížený konstruktor, vytvoří stav označený symbolem q, který má zadané souřadnice pro vykreslení.

Method Summary

java.awt.geom.Point2D.Double	**getcentre**() Vrací hodnotu pozice pro vykreslování stavu automatu.
java.awt.Color	**getcolor**() Vrací hodnotu barvy pro vykreslování stavu automatu.

java.lang.String	**getq**() Vrací symbol označení stavu automatu.
boolean	**nodeFocused**(double radius, java.awt.event.MouseEvent e) Metoda zjistí, zda je stav zaměřený myší.
void	**paintNode**(java.awt.Graphics2D g2D, double radius, java.lang.Boolean fin) Vykreslí uzel (stav automatu) na kreslící plátno.
void	**setcentre**(java.awt.geom.Point2D.Double centre) Nastaví atribut souřadnice pro vykreslování na hodnotu centre.
void	**setcolor**(java.awt.Color color) Nastaví atribut barva pro vykreslování na hodnotu color.
void	**setq**(java.lang.String q) Nastaví hodnotu atributu pro označení stavu na hodnotu q.

B.9 Class PDArule

java.lang.Object
  └ **cfgpdaconv.PDArule**

Třída PDArule, její instance představuje přechodovou relaci zásobníkového automatu podle definice (q1,a,Z)={(q1,α1) ... (qn,αn)}. Objekt přechodové relace ukládá symboly levé strany (q1, a, Z) a referenci na objekty pravých stran (q1,α1) ... (qn,αn), které jsou objekty třídy PDAruleTrans. Konstruktor vytvoří přechodovou relaci. Naimplementována je metoda pro parsování přechodové relace (symboly levé a pravé strany) z řetězce znaků.

Constructor Summary

PDArule() Konstruktor, vytvoří prázdou přechodovou relaci.
PDArule(java.lang.String str) Přetížený konstruktor, vytvoří přechodovou relaci parsováním z řetězce znaků str.
PDArule(java.lang.String stFrom, java.lang.String in, java.lang.String pd, java.lang.String stTo,

java.lang.String pdString) Přetížený konstruktor, vytvoří přechodovou relaci ze zadaných parametrů.	

Method Summary

void	**addTrans**(java.lang.String stTo, java.lang.String pdString) Metoda přidává pravou stranu přechodové relace do množiny objektů pravých stran přechodových relací.
PDAruleTrans	**focusedEdge**(double radius, java.util.Vector q, java.awt.event.MouseEvent e) Metoda vrací zaměřenou hranu (přechodovou relaci automatu) kurzorem myši.
java.lang.String	**getInputSymbol**() Vrací symbol přechodové relace, který představuje symbol čtený ze vstupu při přechodu automatu.
java.lang.String	**getPushdownSymbol**() Vrací symbol přechodové relace, který představuje symbol nacházející se na vrcholu zásobníku při přechodu automatu.
java.lang.String	**getState**() Vrací vstupní stav přechodové relace.
java.util.Vector	**getTrans**() Vrací množinu objektů, které představují pravé strany přechodové relace.
PDAruleTrans	**getTrans**(int i) Vrací objekt, který představuje pravou stranu přechodové relace a nachází se v množině objektů pravých stran na pozici i.
void	**paintEdge**(java.awt.Graphics2D g2D, double radius, java.util.Vector q) Metoda vykreslí přechodovou relaci (hrana grafu) na kreslící plátno.
int	**poz**(java.lang.String stTo) Metoda přiděluje přechodové relaci pozici pro vykreslování.
java.lang.Boolean	**ruleEmpty**() Metoda zjišťuje, zda je přechodová relace nadefinovaná (existují její symboly) nebo je prázdná.

java.lang.Boolean	**ruleEqual**(java.lang.String stFrom, java.lang.String pd) Metoda pro zjištění existence přechodové relace podle zadaných parametrů.
java.lang.Boolean	**ruleEqual**(java.lang.String stFrom, java.lang.String in, java.lang.String pd) Metoda pro zjištění existence přechodové relace podle zadaných parametrů.
java.lang.String	**toString**() Metoda převede objekt přechodové relace na jeho řetězcovou reprezentaci.

B.10 Class PDAruleTrans

java.lang.Object
  └ **cfgpdaconv.PDAruleTrans**

Třída PDAruleTrans, její instance představuje pravou stranu přechodové relace zásobníkového automatu podle definice (q1,α). Objekt ukládá symboly pravé strany (q1,α). Konstruktor vytvoří pravou stranu přechodové relace. Naimplementována je metoda pro parsování pravé strany přechodové relace z řetězce znaků. Jsou zde metody pro kseslení přechodové relace na kreslící plátno a pro zjišťování zaměření myší.

Constructor Summary

PDAruleTrans() Konstruktor, vytvoří prázdou pravou stranu přechodové relace.
PDAruleTrans(java.lang.String str, int poz, java.awt.Color color) Přetížený konstruktor, vytvoří pravou stranu přechodové relace parsováním z řetězce znaků str.
PDAruleTrans(java.lang.String stTo, java.lang.String pdString, int poz, java.awt.Color color) Přetížený konstruktor, vytvoří pravou stranu přechodové relace ze zadaných parametrů.

Method Summary

boolean	**focused**(double radius, java.awt.geom.Point2D.Double odkud, java.util.Vector q, java.lang.String in, java.lang.String pd, java.awt.event.MouseEvent e)

	Metoda zjistí, zda je přechodová relace zaměřená myší.
java.awt.Color	**getColor**() Vrací barvu přechodové relace pro vykreslování.
java.lang.String	**getpdString**() Vrací řetězec znaků, který se ukládá na zásobník při přechodu automatu.
int	**getPoz**() Vrací pozici přechodové relace pro vykreslování.
java.lang.String	**getState**() Vrací stav přechodové relace, do kterého se přechází.
void	**paintEdge**(java.awt.Graphics2D g2D, double radius, java.awt.geom.Point2D.Double odkud, java.util.Vector q, java.lang.String in, java.lang.String pd) Metoda vykreslí přechodovou relaci na kreslící plátno.
void	**setColor**(java.awt.Color color) Nastaví atribut barvy pro vykreslování na hodnotu parametru color
void	**setPoz**(int poz) Nastaví atribut pozice pro vykreslování na hodnotu parametru poz
java.lang.String	**toString**() Metoda převede objekt pravé strany přechodové relace na jeho řetězcovou reprezentaci.

B.11 Class Conversion

```
java.lang.Object
  └ cfgpdaconv.Conversion
```

Třída Conversion, její instance představuje objekt pro provádění transformací bezkontextových gramatik a pro převod bezkontextové gramatiky na zásobníkový automat a naopak. Jsou naimplementovány metody pro:

- transformaci bezkontextové gramatiky na bezkontextovou gramatiku bez nedosažitelných symbolů,
- transformaci bezkontextové gramatiky na bezkontextovou gramatiku bez nepoužitelných symbolů,
- transformaci bezkontextové gramatiky na bezkontextovou gramatiku bez eps-pravidel,
- transformaci bezkontextové gramatiky na bezkontextovou gramatiku bez jednoduchých pravidel,
- transformaci bezkontextové gramatiky na bezkontextovou gramatiku vlastní,
- převod bezkontextové gramatiky na zásobníkový automat,

- převod zásobníkového automatu na bezkontextovou gramatiku.

Constructor Summary

Conversion() Konstruktor, vytvoří objekt pro provádění transformací bezkontextových gramatik a pro převod bezkontextové gramatiky na zásobníkový automat a naopak.	

Method Summary

CFGrammar	**convToCFG**(PDAutomaton pda) Metoda pro převod zásobníkového automatu pda na bezkontextovou gramatiku ekvivalentní vstupnímu zásobníkovému automatu.
CFGrammar	**convToEpsilonFreeGrammar**(CFGrammar cfg) Metoda pro transformaci vstupní bezkontextové gramatiky cfg na ekvivalentní bezkontextovou gramatiku bez eps-pravidel.
CFGrammar	**convToGrammarWithoutInaccessibleSymbols**(CFGrammar cfg) Metoda pro transformaci vstupní bezkontextové gramatiky cfg na ekvivalentní bezkontextovou gramatiku bez nedosažitelných symbolů.
CFGrammar	**convToGrammarWithoutNontermSymbols**(CFGrammar cfg) Metoda pro transformaci vstupní bezkontextové gramatiky cfg na ekvivalentní bezkontextovou gramatiku bez neterminálních symbolů, které nemohou vygenerovat terminální řetězec.
CFGrammar	**convToGrammarWithoutUselesSymbols**(CFGrammar cfg) Metoda pro transformaci vstupní bezkontextové gramatiky cfg na ekvivalentní bezkontextovou gramatiku bez nepoužitelných symbolů.
PDAutomaton	**convToPDA**(CFGrammar cfg) Metoda pro převod bezkontextové gramatiky cfg na zásobníkový automat pda ekvivalentní vstupní bezkontextové gramatice.
CFGrammar	**convToProperGrammar**(CFGrammar cfg) Metoda pro transformaci vstupní bezkontextové gramatiky cfg na ekvivalentní vlastní bezkontextovou gramatiku.
CFGrammar	**convToUnitFreeGrammar**(CFGrammar cfg) Metoda pro transformaci vstupní bezkontextové gramatiky cfg na ekvivalentní bezkontextovou gramatiku bez jednoduchých pravidel.

C Obsah přiloženého CD

K diplomové práci je přiložen CD-ROM obsahující zdrojové kódy programu, soubory programové dokumentace, elektronickou verzi této diplomové práce a zdrojový text HTML stránky, na kterou je umístěn naprogramovaný Java applet. V kořenovém adresáři je soubor `readme.txt`, který obsahuje popis obsahu CD.

C.1 Struktura adresářů

- `source` - V tomto adresáři jsou umístěny zdrojové kódy programu v jazyce Java.
- `doc` - Tento adresář obsahuje dokumentaci zdrojových kódů ve formátu Javadoc, tedy ve formě HTML stránek.
- `pdf` - Zde je umístěna elektronická verze této diplomové práce.
- `applet`- V tomto adresáři jsou umístěny soubory potřebné pro zobrazení HTML stránky s naprogramovaným Java appletem.

C.2 Spuštění programu

Aplikace je naprogramována jako Java applet. Pro spuštění zobrazte v HTML prohlížeči soubor `index.html` z adresáře `applet` umístěného na CD. Alternativou je zobrazení stránky www.convertcfg.php5.cz z veřejné internetové sítě.

Stránku lze prohlížet pomocí libovolného prohlížeče s rozšířením pro Java 2. Důležitou podmínkou správného zobrazení stránky je, aby prohlížeč spouštění appletů povoloval.

Pokud se applet na stránce správně nezobrazí, zkontrolujte, zda je spouštění appletů povoleno. Případně je nutné nainstalovat Java Plug-in (J2SE JRE nebo JDK) ze stránek http://java.sun.com/javase/downloads/index.jsp. Nainstalujte poslední verzi, případně 1.4.2 nebo vyšší.