MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

WEBOVÝ FORMULÁŘ PRO VÝPOČET FUNKCÍ FIRST A FOLLOW

Bakalářská práce

Michal Hanzelka

Brno, 2006

Prohlášení

Prohlašuji, že tato práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

V Brně dne 6. ledna 2006 .....

Poděkování

Na tomto místě bych chtěl poděkovat Jiřímu Barnatovi za vedení a trpělivost, Ondrovi Výbornému za věcné připomínky a dalším lidem, kteří mě podporovali.

Shrnutí

Práce řeší problematiku výpočtu funkcí FIRST a FOLLOW nad bezkotextovými gramatikami a rozhodování o zadané gramatice, zda je SLL(k).

V první části práce jsou definovány základní pojmy a popsány principy použitých algoritmů.

Druhá část práce popisuje strukturu programu a ovládání vytvořené aplikace.

Klíčová slova

Bezkontextová gramatika, redukovaná gramatika, funkce FIRST, funkce FOLLOW, silná LL(k) gramatika, Java aplet

OBSAH

1 Úvod

Během studia informatiky se každý dříve či později setká s teorií formálních jazyků. V informatickém smyslu znamená pojem formální jazyk množinu slov nad určitou abecedou. Tuto často nekonečnou množinu lze reprezentovat různými způsoby. Jedním z nich jsou gramatiky, jejichž zkoumání nám přináší důležité poznatky o struktuře a vlastnostech jazyků, které generují. Mezi nástroje, které k tomuto zkoumání můžeme použít, patří funkce FIRST a FOLLOW, které slouží ke snadnějšímu rozpoznání a pochopení vztahů uvnitř těchto gramatik. Jejich použitím lze definovat například množinu SLL(k) gramatik, které se využívají ke konstrukci programovacích jazyků. Výhodou tohoto typu gramatik je jejich jednoduchost a tedy i vyšší rychlost následné syntaktické analýzy, která je s výhodou uplatněna při překladu programů napsaných v jazyce založeném na těchto gramatikách.

Cílem této bakalářské práce je vytvořit na základě poznatků z teorie formálních jazyků aplikaci, která by umožňovala počítat nad zadanou bezkontextovou gramatikou funkce FIRST a FOLLOW a rozhodovala by o vstupní gramatice, zda je či není SLL(k). Záměrem bylo koncipovat program jako webový formulář, který by kromě výše zmíněných funkcí navíc umožňoval pracovat s textovými soubory obsahujících gramatiky.

Aplikace může posloužit studentům k hlubšímu porozumění této abstraktní a tudíž poměrně obtížné kapitoly teoretické informatiky, popřípadě k praktickému ověření dříve získaných poznatků.

2 Použité pojmy

- *Gramatika* – čtveřice (N, Σ, P, S), kde N je konečná množina terminálních symbolů (často označovaných jako terminály), Σ je konečná množina neterminálních symbolů (neterminálů), P je konečná množina pravidel a S je počáteční neterminál $(S \in N)$.

- *Bezkontextová gramatika* – gramatika, která má všechna pravidla ve tvaru $X \to \alpha$, kde X je neterminál z N a α je konečná posloupnost terminálů a neterminálů.

- *Redukovaná gramatika* – gramatika je redukovaná právě tehdy, když z každého jejího neterminálu lze odvodit terminální řetězec (tedy řetězec složený jen z terminálů) a z počátečního neterminálu lze odvodit všechny ostatní neterminály.

- *Funkce* $FIRST(\alpha)_k^G$ – je množina terminálních řetězců délky maximálně k, jimiž mohou začínat řetězce derivované z α podle pravidel gramatiky G, kde α je libovolný řetězec generovaný gramatikou G (formální definice funkce FIRST je uvedena v části 3.2).

- *Funkce* $FOLLOW(X)_k^G$ – je množina terminálních řetězců délky maximálně k, které se mohou vyskytovat bezprostředně vpravo od neterminálu X v libovolném odvození gramatiky G (formální definice funkce FOLLOW je uvedena v části 3.3).

3 Definice a algoritmy

Program vytvořený v rámci této práce je založen na použití čtyř hlavních algoritmů, jejichž principy budou vysvětleny v následujících podkapitolách.

Definice funkcí FIRST a FOLLOW a silné LL(k) gramatiky, uvedené v této kapitole, jsou převzaty z (2, strany 1 a 4).

3.1 Redukce gramatiky

Náročnost zpracování gramatik rychle roste v závislosti na jejich složitosti. Proto se snažíme gramatiky co nejvíce zjednodušit a odstranit nadbytečné části (neterminály, terminály, pravidla) při zachování generovaného jazyka. Odstranění těchto částí se nazývá *redukce gramatiky*.

Redukovat gramatiku znamená odstranit z ní všechny nepoužitelné symboly. Rozeznáváme dva druhy nepoužitelných symbolů. Jedná se o *nedosažitelné symboly*, tedy o neterminály a terminály, které se neobjeví v žádném odvození z počátečního neterminálu na terminální řetězec (nejsou součástí žádného použitého pravidla) a o neterminály, ze kterých nelze odvodit žádný terminální řetězec (*nepoužitelné neterminály*).

Následující tři algoritmy, které dohromady provádějící redukci gramatiky, jsou převzaté z (1, strany 62-63) a mírně upraveny.

3.1.1 Popis algoritmu

Algoritmus redukující gramatiku se skládá ze dvou hlavních kroků. V prvním z nich se odstraní z gramatiky všechny *nepoužitelné neterminály* a ve druhém se odstraní *nedosažitelné symboly*.

Použitím algoritmu 3.1 se ze vstupní gramatiky odstraní všechny *nepoužitelné neterminály* a všechna pravidla, ve kterých se tyto neterminály objevují.

Vstup : *bezkontextová gramatika* $G = (N, \Sigma, P, S)$
Výstup : *bezkontextová gramatika* $G' = (N', \Sigma, P', S)$ *bez neterminálů, ze kterých nelze odvodit terminální řetězec* : $L(G) = L(G')$
Dle následujících kroků konstruuj induktivně množiny $N_0, N_1, \ldots$ *takto* :

(1) $i = 0; N_0 = \varnothing;$
repeat
(2) $i = i + 1;$
(3) $N_i = N_{i-1} \cup \{A \mid A \to \alpha \in P, \alpha \in (N_{i-1} \cup \Sigma)^*\};$
(4) *until* $N_i == N_{i-1}$
(5) $N' = N_i;$
(6) $P' = P \cap (N' \times (N' \cup \Sigma)^*\};$
(7) $G' = (N', \Sigma, P', S);$

Algoritmus 3.1: Redukce nepoužitelných neterminálů

V kroku (1) algoritmu 3.1 proběhne inicializace – do proměnné *i* se uloží hodnota nula a do N_o prázdná množina. Krok (2) zvýší hodnotu proměnné *i* o jedna. V kroku (3) se do proměnné N_i vloží neterminály z N_{i-1} a neterminály, které stojí na levé straně některého pravidla z *P*, jenž má na pravé straně řetězec složený pouze z terminálů a neterminálů z proměnné N_{i-1} nebo prázdné slovo. Kroky (2) a (3) se opakují tak dlouho, dokud jsou proměnné N_i a N_{i-1} různé, tedy dokud není splněna podmínka (4). V dalších krocích se do N' vloží obsah proměnné N_i a do P' všechna pravidla z *P*, která mají na levé straně neterminál obsažený v N' a na pravé straně řetězec složený z terminálů a neterminálů obsažených v proměnné N' nebo prázdné slovo. Tím se do P' nedostanou pravidla obsahující *nepoužitelné neterminály*. V posledním kroku (7) je vytvořena výstupní gramatika.

Algoritmus 3.2, jenž představuje druhý krok redukce gramatiky, odstraní ze vstupní gramatiky *nedosažitelné symboly* a pravidla, ve kterých se vyskytují.

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

Výstup : *bezkontextová gramatika* $G' = (N', \Sigma', P', S)$ *bez nedosažitelných symbolů* :
$L(G) = L(G')$

Dle následujících kroků konstruuj induktivně množiny $V_0, V_1, \ldots$ *takto* :

(1) $i = 0; V_0 = \{S\};$
repeat
(2) $i = i + 1;$
(3) $V_i = V_{i-1} \cup \{X \mid \exists A.(A \to \alpha X \beta \in P \wedge A \in V_{i-1})\};$
(4) *until* $V_i == V_{i-1};$
(5) $N' = N \cap V_i;$
(6) $\Sigma' = \Sigma \cap V_i;$
(7) $P' = P \cap (V_i \times V_i^*);$
(8) $G' = (N', \Sigma', P', S);$

Algoritmus 3.2: Redukce nedosažitelných symbolů

Při provádění algoritmu 3.2 je v proměnné i uložena hodnota odpovídající počtu již provedených průchodů cyklu repeat-until. Proměnná V_i obsahuje symboly, které jsou dosažitelné z počátečního neterminálu v nejvýše i krocích odvození. V prvním kroku algoritmu 3.2 je provedena inicializace proměnných. V kroku (2) se zvýší hodnota i o jedna. Krokem (3) se do proměnné V_i vloží hodnota proměnné V_{i-1} a dále všechny symboly, které jsou součástí pravé strany takového pravidla z P, ve kterém na levé straně stojí některý neterminál z V_{i-1}. Kroky (2) a (3) se opakují tak dlouho, dokud jsou proměnné V_i a V_{i-1} různé, tedy dokud přidáváme nové symboly do V_i. Ve zbývajících krocích je vytvořena výstupní gramatika, které obsahuje pouze symboly dosažitelné z počátečního neterminálu, pravidla složená pouze z těchto symbolů a původní počáteční neterminál.

Algoritmus 3.3 provádějící redukci bezkontextové gramatiky používá postupně dva výše popsané algoritmy.

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

Výstup : *redukovaná bezkontextová gramatika* $G' = (N', \Sigma', P', S) : L(G) = L(G')$

(1) *Použij algoritmus 3.1 se vstupem* G *a výstupem* G_1.

(2) *Použij algoritmus 3.2 se vstupem* G_1 *a výstupem* G_2.

(3) $G' = G_2$

Algoritmus 3.3: Redukce gramatiky

3.2 Definice a výpočet funkce FIRST

Výslednou hodnotou funkce $\text{FIRST}(\alpha)_k^G$ je množina všech možných terminální řetězců délky maximálně k, jimiž mohou začínat řetězce odvozené z α v gramatice G.

Formální definice funkce FIRST:

Nechť $G = (N, \Sigma, P, S)$ *je bezkontextová gramatika a* k *je celé kladné číslo.*
Funkci $FIRST_k^G : (N \cup \Sigma)^+ \rightarrow \{w \in \Sigma^* \mid |w| \leq k\}$ *definujeme předpisem*
$FIRST_k^G(\alpha) = \{w \in \Sigma^* \mid (\alpha \rightarrow^* w \wedge |w| \leq k) \vee (\alpha \rightarrow^* wu \wedge |w| = k; u \in \Sigma^*)\}$

Definice 3.1: Funkce FIRST

Před uvedením algoritmu pro výpočet funkce FIRST je ještě nutno definovat funkci „*zřetězení do délky nejvýše k*“. Definice 3.2 obsahuje definici této funkce postupně pro řetězce a pro množiny řetězců.

Nechť $w = a_1a_2...a_n$ *je libovolný řetězec a* k *je celé kladné číslo.*
Pak definujeme funkci $\odot_k : \Sigma^* \rightarrow \Sigma^*$ *takto :*

$$\odot_k w = \begin{cases} a_1...a_k & \text{pro } k<n \\ w & \text{pro } k \leq n \end{cases}$$

Nechť Σ *je abeceda,* $L_1, L_2 \in \Sigma^*$, $k \geq 1$.
Funkci $\oplus_k : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$
definujeme takto :

$$L_1 \oplus_k L_2 = \{w \mid w = \odot_k(x \cdot y) \text{ pro nějaká } x \in L_1, y \in L_2\}$$

Definice 3.2: Zřetězení do délky nejvýše k

3.2.1 Popis algoritmu

V algoritmu 3.4 je pro $FIRST(\alpha)_k^G$ použito zkrácené označení $FI(\alpha)_k$. Tento algoritmus byl s drobnými úpravami převzat z (2, strany 1-2).

Je dána gramatika $G = (N, \Sigma, P, S)$ *a řetězec* $\alpha = Y_1 \cdot Y_2 \cdots Y_m$, *kde* $Y_x \in (N \cup \Sigma)$.

(1) $Fl_k(x) = \{x\}$ *pro* $x \in \Sigma \vee x = \varepsilon$

(2) *Výpočet* $Fl_k(x)$ *pro* $x \in N$:

Nechť $N = \{X_1, X_2, ..., X_n\}$. *Budeme počítat hodnotu* $Fl_k(X_i)$ *současně pro všechny neterminály* $(i = 1,..,n)$. *Nechť všechna pravidla pro neterminál* X_i *jsou tato*:

$X_i \to Y_1^1 \cdots Y_{k_1}^1 \mid Y_1^2 \cdots Y_{k_2}^2 \mid \cdots \mid Y_1^j \cdots Y_{k_j}^j$

Potom

(2.1) *Počáteční hodnoty jsou* $Fl_k(X_i) = \varnothing$.

(2.2) $Fl_k(X_i) = [Fl_k(Y_1^1) \oplus_k Fl_k(Y_2^1) \oplus_k ... \oplus_k Fl_k(Y_{k_1}^1)]$

$\cup ... \cup$

$[Fl_k(Y_1^j) \oplus_k Fl_k(Y_2^j) \oplus_k ... \oplus_k Fl_k(Y_{k_j}^j)]$.

(2.3) *Hodnoty* $Fl_k(X_i)$ *jsou pevnými body uvedené soustavy rekurzivních rovnic.*

(3) $FIRST_k(\alpha) = Fl_k(Y_1) \oplus_k Fl_k(Y_2) \oplus_k ... \oplus_k Fl_k(Y_l)$

Algoritmus 3.4: Výpočet funkce FIRST

Abychom zjistili hodnotu funkce FIRST pro vstupní řetězec α, musíme nejdříve vypočítat hodnoty funkce FIRST pro jednotlivé symboly tohoto řetězce. Krok (1) říká, že pokud je argumentem funkce FIRST terminál nebo epsilon, obsahuje výsledná množina řetězec skládající se pouze z tohoto terminálu, respektive znaku epsilon.

Je-li argumentem funkce FIRST neterminál, vypočítáme její hodnotu ve dvou krocích. Krokem (2.1) pro každý neterminál X z N vytvoříme množinu $FI_k(X)$, která je na počátku výpočtu prázdná. V kroku (2.2) postupně procházíme všechny neterminály X a pro každé pravidlo, ve kterém se X vyskytuje na levé straně, vložíme do množiny $FI_k(X)$ řetězce délky maximálně k, které vzniknou zřetězením $FI_k(Y_1) \oplus_k \cdots \oplus_k FI_k(Y_j)$, kde $Y_1 \cdots Y_j$ je řetězec na pravé straně pravidla. Krok (2.2) aplikujeme tak dlouho, dokud lze do některé množiny $FI_k(X)$ přidat další symbol.

Hodnotu funkce FIRST pro vstupní řetězec α vypočteme *„zřetězením do délky nejvýše k“* množin $FI_k(Y_1), ..., FI_k(Y_m)$.

3.3 Definice a výpočet funkce FOLLOW

Výsledkem funkce $\mathrm{FOLLOW(X)}_k^G$ je množina, jenž obsahuje všechny terminální řetězce délky nejvýše k, které se mohou objevit bezprostředně vpravo do neterminálu X v nějakém odvození z počátečního neterminálu gramatiky G.

Formální definice funkce FOLLOW:

Nechť $G = (N, \Sigma, P, S)$ *je bezkontextová gramatika a* k *je celé kladné číslo.*

Funkci $FOLLOW_k^G : N \to \{w \in \Sigma^* \mid |w| \le k\}$ *definujeme předpisem*

$$FOLLOW_k^G(X) = \{w \in \Sigma^* \mid (S \to^* \gamma X \alpha, w \in FIRST_k^G(\alpha); \gamma \in \Sigma^*; \alpha \in (\Sigma \cup N)^*\}$$

Definice 3.3: Funkce FOLLOW

3.3.1 Popis algoritmu

Algoritmus 3.5 byl s drobnými úpravami převzat z (2, strana 2). Pro označení funkce $\mathrm{FOLLOW(N)}_k^G$ se v něm používá zkrácený zápis $\mathrm{FO(N)}_k$ a funkce $\oplus_k$ je definována stejně jako v části 3.2.

Je dána gramatika $G = (N, \Sigma, P, S)$. *Funkce FO je definována pro* $X \in N$.

(1) *Postupně počítáme hodnoty* :

$FO_1(X)$ *pro všechny* $X \in N$,

$FO_2(X)$ *pro všechny* $X \in N$,

$\vdots$

$FO_k(X)$ *pro všechny* $X \in N$,

(2) *Při výpočtu* FO_i *postupujeme následovně* :

(2.1) $FO_i(S) = \{\varepsilon\}$ *pro počáteční neterminál* S.

$FO_i(X) = \varnothing$ *pro ostatní neterminály*.

(2.2) *Pro každé pravidlo tvaru* : $Y \to \alpha X \beta \in P$, *kde* $\beta \neq \varepsilon$

$FO_i(X) = FO_i(X) \cup [(FI_i(\beta) - \{\varepsilon\}) \oplus_i FO_{i-1}(Y)]$

(2.3) *OPAKUJ*

Pro každé pravidlo tvaru : $Y \to \alpha X \beta \in P$, *kde* $\beta = \varepsilon$ *nebo* $\varepsilon \in FI_1(\beta)$

$FO_i(X) = FO_i(X) \cup FO_i(Y)$

DOKUD lze do některé množiny přidat další symbol

Algoritmus 3.5: Výpočet funkce FOLLOW

Při výpočtu funkce FOLLOW počítáme postupně množiny FO_1 až FO_k pro všechny neterminály gramatiky. Výpočet množin FO_i se skládá ze tří kroků, přičemž třetí krok se provádí tak dlouho, dokud lze do některé množiny FO_i přidat další symbol. Krok (2.1): všechny množiny $FO_i(X)$ jsou prázdné kromě $FO_i(S)$, která obsahuje prázdné slovo. Krok (2.2): pro každé pravidlo tvaru $Y \to \alpha X \beta$ přidáme do množiny $FO_i(X)$ všechny prvky množiny $FI_i(\beta)$ bez ε, zřetězené do délky nejvýše i s množinou $FO_{i-1}(Y)$. Tímto krokem se do množiny $FO_i(X)$ dostanou všechny terminální řetězce odvoditelné ze symbolů, které jsou v některém pravidle bezprostředně vpravo od X. Krok (2.3): pro každé pravidlo $Y \to \alpha X$ nebo $Y \to \alpha X \beta$, kde $FI_i(\beta)$ obsahuje ε, přidáme do množiny $FO_i(X)$ všechny prvky množiny $FO_i(Y)$. Pokud se v pravidle $Y \to \alpha X \beta$ bezprostředně vpravo od X může vyskytnou prázdný řetězec, vloží se do množiny $FO_i(X)$ i ty terminální řetězce, které se mohou objevit bezprostředně vpravo od neterminálu Y.

3.4 Definice silných LL(k) gramatik

Je-li gramatika SLL(k), pak číslo *k* udává počet nejlevějších znaků terminálního řetězce, které musíme „vidět“, abychom mohli deterministicky určit pravidlo, jenž bylo použito k odvození tohoto řetězce.

Formální definice silné LL(k) gramatiky:

> *Nechť G je redukovaná bezkontextová gramatika. G je silná LL(k) gramatika, právě když pro jakákoliv dvě pravidla se stejnou levou stranou* $A \to \alpha$, $A \to \beta$ *kde* $\alpha \neq \beta$, *platí* :
>
> $$FIRST_k^G(\alpha \cdot FOLLOW_k^G(A)) \cap FIRST_k^G(\beta \cdot FOLLOW_k^G(A)) = \emptyset$$

Definice 3.4: Silná LL(k) gramatika

Algoritmus 3.6, který rozhoduje, zda je gramatika SLL(k), využívá dříve definovaných funkcí FIRST a FOLLOW.

Vstup: *bezkontextová gramatika* $G = (N, \Sigma, P, S)$, *celé číslo* $k > 0$
Výstup: "*Ano*" *je-li G SLL(k) gramatika*, "*Ne*" *v opačném případě*

je_SLL = true;
pro každý neterminál A z N do {
  pro každé dvě pravidla $A \to \alpha$, $A \to \beta$, *kde* $\alpha \neq \beta$ *do* {
    if $(FIRST_k^G(\alpha \cdot FOLLOW_k^G(A)) \cap FIRST_k^G(\beta \cdot FOLLOW_k^G(A)) \neq \emptyset)$
      then je_SLL = false;
  }
}
if (*je_SLL == true*) *then Výstup* = "*Ano*" *else Výstup* = "*Ne*"

Algoritmus 3.6: Je gramatika SLL(k)?

4 Struktura programu

4.1 Volba programovacího jazyka a vývojového prostředí

Program jsem se rozhodl napsat v jazyce Java. Jednou z výhod tohoto v současnosti velmi populárního jazyka je jeho platformová nezávislost. Programy, které jsou v něm napsány, nejsou vázány na použití konkrétního operačního systému a lze je spustit ve všech operačních systémech, které jazyk Java podporují. Java také přináší možnost vkládat pomocí Java apletů jednoduché programy do HTML kódu webové stránky a spouštět je tak přímo v internetovém prohlížeči. Tyto dva aspekty umožňují, aby byl výsledný program snadno dostupný přímo na Internetu.

Další výhodou tohoto jazyka je jeho široká programová podpora. Na Internetu je volně ke stažení mnoho kvalitních vývojových prostředí, která velmi usnadňují práci programátorům.

Pro tvorbu výsledné aplikace jsem použil nejnovější (prosince 2005) verzi jazyka Java – Java SDK 1.5[1] (podle dřívějšího značení J2SE JDK 1.5), která oproti předchozí verzi umožňuje pracovat s generickými typy a zavádí novou syntax cyklu `for`, jejichž použití činí program přehlednějším a bezpečnějším.

Třídy balíčku *dat_typy* a jejich odladění jsem provedl v prostředí NetBeans IDE 4.0. Pro konečné úpravy a pro vytvoření grafického rozhraní, tedy zejména třídy *FormularApplet*, jsem využil program JBuilder 2005.

4.2 Reprezentace gramatik

Pro reprezentaci symbolů, řetězců symbolů a gramatik jsem navrhl celkem tři objektové datové struktury. Ty jsem implementoval ve třídách *Znak, Retezec* a *Gramatika*. Dále jsem vytvořil několik druhů výjimek, které mohou být vyvolány metodami výše zmíněných tříd. Všechny tři strukturní třídy spolu s výjimkami tvoří samostatně distribuovatelný balíček *webovyformular.dat_typy*.

Ve třídách *Znak* a *Gramatika* jsou překryty následující metody třídy *Object*, která je implicitním předkem všech nově vytvořených tříd:

- *toString()* - vrací na výstup řetězec *String*, jenž je znakovou reprezentací dané instance. Ve třídě *Znak* vrací atribut *jmeno,* ve třídě *Retezec* posloupnost jmen symbolů objevujících se v daném *Retezci*. Ve třídě *Gramatika* vrací řetězec obsahující množinu neterminálů, množinu terminálů, jméno počátečního neterminálu a jednotlivá pravidla, formátovaná do běžně používané podoby.

[1] Kompletní oficiální název zní Java 2 Standart Edition™ Development Kit 5.0 Update 5

- *hashCode()* – metoda vrací na výstup hešovací kód pro danou instanci. K vytvoření hešovacího kódu jsou použity atributy volající instance a metody *hashCode* elementárních datových typů. Takto vytvořený kód by měl být pro každou instanci „relativně unikátní“.

- *equals(o)* - vrací hodnotu *true* pokud jsou volající instance a objekt *o* shodní. V opačném případě vrací *false*.

Pro vytvoření všech metod *hashCode()* a *equals()* jsem použil některá doporučení uvedená v (5, strany 166-168 a 159-161).

4.2.1 Třída Znak

Instance třídy *Znak* reprezentuje právě jeden symbol konkrétní gramatiky. Symbolem se v tomto smyslu rozumí buď terminál, neterminál nebo prázdné slovo – epsilon[2].

```
public class Znak {
  public static String jmenoEps = "eps";
  private String jmeno;
  private byte typ;
...
```

Zdrojový kód 4.1: Deklarační hlavička třídy Znak

Atribut *jmeno* typu *String* obsahuje název symbolu, jenž by měl být v dané gramatice unikátní. Jméno epsilonu je uloženo ve veřejné statické proměnné *jmenoEps,* která implicitně obsahuje řetězec „*eps*“.

Atribut *typ* datového typu *byte* určuje, o jaký druh symbolu se jedná. Pro terminál je jeho hodnota „1“, pro neterminál „2“ a pro epsilon „0“. Třída obsahuje metody *isNeterminal, isTerminal* a *isEpsilon*, které vracejí hodnotu typu *boolean*, jenž vyjadřuje, zda je volající instance odpovídajícího druhu.

[2] Přestože epsilon zastupuje prázdný řetězec (prázdné slovo) a logicky se tedy liší od terminálů a neterminálů, rozhodl jsem se ho z implementačních důvodů také reprezentovat instancí třídy *Znak*.

```
public Znak(String jmeno, byte typ) throws ParamPrazdnyStringException,
    PrefixEpsilonuException {
  if (!jmeno.trim().equals("")) {
    if (!jmenoEps.startsWith(jmeno)||(typ == 0)) {
      this.jmeno = ( (typ == 0) ? Znak.jmenoEps : jmeno);
      this.typ = typ;
    }
    else {
      System.err.println("Jméno Znaku " + jmeno + " je prefixem " +
                         jmenoEps + ", což je jméno epsilonu.");
      throw new PrefixEpsilonuException();
    }
  }
  else {
    System.err.println("Jméno Znaku nemůže být prázdný řetězec.");
    throw new ParamPrazdnyStringException();
  }
}
```

Zdrojový kód 4.2: Konstruktor třídy Znak

Konstruktor má dva parametry – řetězec reprezentující jméno symbolu a hodnotu typu *byte*, která udává, jaký druh symbolu se má vytvořit. Je-li hodnotou vstupního parametru *jmeno* prázdný řetězec nebo řetězec prázdných znaků, je vyvolána výjimka *ParamPrazdnyStringException* a přiřazení parametrů k atributům neproběhne. V případě, že má vzniknout symbol, který není epsilon, ale jeho jméno je prefixem řetězce uloženého v proměnné *jmenoEps*, konstruktor nedovolí takovýto *Znak* vytvořit a vyvolá patřičnou výjimku.

4.2.2 Třída Gramatika

Jak již název napovídá, instance třídy *Gramatika* představuje jednu konkrétní gramatiku. Obsahuje atributy *neterminaly, terminaly, pravidla* a *pocatek*, které reprezentují postupně množinu neterminálních symbolů, množinu terminálních symbolů, množinu pravidel a počáteční neterminál gramatiky.

```
public class Gramatika {
  private HashSet<Znak> neterminaly;
  private HashSet<Znak> terminaly;
  private HashMap<Znak, HashSet<ArrayList<Znak>>> pravidla;
  private Znak pocatek;
  ...
```

Zdrojový kód 4.3: Deklarační hlavička třídy Gramatika

Konstruktor třídy *Gramatika* obsahuje pouze jednoduchá přiřazení vstupních parametrů k atributům:

```
public Gramatika (HashSet<Znak> neterminaly,
                  HashSet<Znak> terminaly,
                  HashMap<Znak, HashSet<ArrayList<Znak>>> pravidla,
                  Znak pocatek) {
   this.neterminaly = neterminaly;
   this.terminaly = terminaly;
   this.pravidla = pravidla;
   this.pocatek = pocatek;
}
```

Zdrojový kód 4.4: Konstruktor třídy Gramatika

Důležitými metodami třídy *Gramatika* jsou *redukujNeterminaly*, *redukuj* a *isSll*. První z nich eliminuje z gramatiky neterminály, ze kterých nelze odvodit terminální řetězec. Tato metoda je volána v těle metody *redukuj*, která z gramatiky odstraní všechny nepoužitelné symboly. Tyto dvě metody implementují algoritmy 3.1 *Redukce nepoužitelných neterminálů* a 3.3 *Redukce gramatiky*, jenž byly popsány v části 3.1 *Redukce gramatiky*.

Metoda *isSll* realizuje algoritmus 3.6 *Je gramatika SLL(k)?* uvedený v části 3.4 a vrací hodnotu *true*, pokud volající instance třídy *Gramatika* splňuje definici SLL(k) gramatik (definice 3.4), kde *k* je vstupním parametrem metody. V opačném případě vrací *false*.

4.2.3 Třída Retezec

Při práci s gramatikami se často používají funkce zpracovávající řetězce symbolů. Proto jsem vytvořil třídu *Retezec*, která je určena pro práci s objekty typu *ArrayList<Znak>*, jenž v objektovém návrhu reprezentují řetězce terminálních a neterminálních symbolů[3]. Třída neobsahuje konstruktor a všechny její metody jsou statické.

[3] V dalším textu je pod pojmem *Retezec* myšlen objekt typu *ArrayList<Znak>*.

Metoda *toRetezecZnaku* převede vstupní řetězec *String* obsahující jména symbolů na odpovídající *Retezec*, který poté vrací na výstup. K rozlišení jednotlivých jmen symbolů je použito prvků množin terminálů a neterminálů, zadaných jako argumenty této metody.

Metoda *zretez* ve své základní podobě řetězí dva *Retezce* zadané na vstupu. V přetížených verzích zřetězuje i množiny *Retezců* a provádí také zřetězení *„do délky nejvýše k“*, jejichž výsledkem jsou *Retezce* délky maximálně *k*.

Důležitou metodou je metoda *first* jenž provádí výpočet množiny FIRST zadaného *Retezce*. Tato metoda implementuje algoritmus 3.4 *Výpočet funkce FIRST* uvedený v části 3.2.1.

Obdobně metoda *follow* použitím algoritmu 3.5 *Výpočet funkce FOLLOW* popsaného v části 3.3.1 vypočítá množinu FOLLOW pro zadaný neterminál. Argumentem této metody je neterminál, takže implementačně by měla patřit spíše do třídy *Znak*. Připadalo mi však logičtější mít dvě souvisejících funkce FIRST a FOLLOW v jedné třídě. Proto jsem se rozhodl umístit i tuto metodu do třídy *Retezec*.

4.3 Třída FormularApplet

Tato třída, jenž je rozšířením třídy *Applet*, vytváří uživatelské prostředí formuláře. Obsahuje, zobrazuje a provádí obsluhu událostí jednotlivých grafických komponent. Do třídy je importován balíček *dat_typy,* jenž tvoří jádro aplikace.

Vzhledem k tomu, že tato třída je převážně grafickou nadstavbou balíčku *dat_typy,* neobsahuje žádné složité metody, který by bylo nutné vysvětlit. Proto v této kapitole popíši jen její hlavní rysy a nebudu se zabývat jejich implementací. Část zdrojového kódu této třídy byl automaticky vygenerován program JBuilder.

4.3.1 Vstupy a výstupy

Aplikace umožňuje kromě zadání gramatiky z klávesnice také vložení dat do formuláře ze souboru. Uživateli jsou k dispozici funkce pro načtení gramatiky ze souboru uloženého na lokálním disku klienta a z databáze gramatik uložené na domovském serveru apletu.

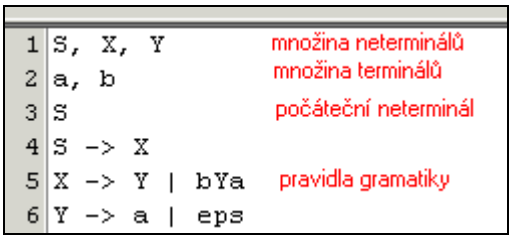
Server, uchovávající databázi gramatik, je kontaktován v inicializační fázi apletu a z příslušného adresáře jsou načtena jména souborů obsahujících gramatiky. Ta jsou poté zobrazena ve výběrovém menu, ve kterém si uživatel může jedno zvolit a po stisknutí tlačítka *Načti z databáze* se obsah zvoleného souboru načte do formuláře.

Načítání souborů z disku klienta je trochu složitější. Z důvodů bezpečnosti totiž aplety nemohou číst, vytvářet a přepisovat soubory v počítači, na kterém je aplet spuštěn. Aby aplet mohl pracovat se soubory na lokálním disku, musí být opatřen digitálním podpisem, který identifikuje původce programu. Uživatel se tak před spuštěním apletu musí rozhodnout, zda důvěřuje osobě, která aplet podepsala a pokud v nabídce zvolí že nikoliv, aplet se spustí pouze s omezenými právy, což mu nedovolí pracovat se soubory na disku klienta (tedy v našem případě načítat gramatiky ze souborů do formuláře a ukládat gramatiky z formuláře do souborů).

Při načítání gramatiky ze souboru, uloženého na lokálním disku klienta, je použit objekt třídy *FileDialog*. Tento objekt zobrazí dialogové okno, ve kterém si uživatel zvolí soubor, jenž chce do formuláře načíst. Pro toto okno je v programu vytvořen filtr souborů (ob-

jekt *FileNameFilter*) s přednastavenou maskou **.txt*, který by měl způsobit, že se v nabídce objeví pouze soubory s touto příponou. Bohužel, v souladu s tím co je napsáno v (6), v operačních systémech Windows filtr jmen souborů nefunguje a uživatel tak může vybrat jakýkoliv soubor. Načítání obsahu souboru probíhá po řádcích, které se postupně vkládají do komponent formuláře v pořadí *Množina neterminálů*, *Množina terminálů*, *Počáteční neterminál* a zbývající řádky souboru se vloží do textového pole *Množina Pravidel*. Příklad gramatiky zapsané v textovém souboru ve správném formátu je na obrázku 4.1.

Ukládání gramatiky z formuláře do souboru na disku klienta probíhá podobně jako načítání souboru. Zobrazí se okno pro výběr souboru (v systémech Windows opět nefunguje filtr jmen souborů) a uživatel si může buď vybrat soubor již existující nebo zadat jméno nového souboru. Do něho se zapíše aktuální obsah komponent formuláře tvořící gramatiku. Na každý řádek zvlášť se postupně vloží obsahy vstupních polí *Množina neterminálů*, *Množina terminálů*, *Počáteční neterminál* a za ně na další řádky obsah textového pole *Množina pravidel* včetně znaků ukončující řádky (viz obrázek 4.1).



Obrázek 4.1: Gramatika uložená v textovém souboru

4.3.2 Tlačítka FIRST(k), FOLLOW(k) a SLL(k)

Protože funkce FIRST a FOLLOW jsou definovány pouze pro bezkontextové gramatiky, musí program umět rozeznat, zda gramatika zadaná na vstupu splňuje toto kritérium. K tomu slouží podmínky kladené uživateli na volbu jmen neterminálů a terminálů a přesně definovaný způsob zadávání pravidel gramatiky. Pro všechna jména musí platit, že nejsou prefixem (předponou) žádného jiného jména, ať už terminálu nebo neterminálu. Navíc je při konverzi obsahu vstupních polí na objekt typu *Gramatika* očekáván na levé straně každého řádku textového pole *Pravidla* právě jeden neterminál z *Množiny neterminálů*. Pokud tomu tak není, je vyvoláno chybové hlášení, které na tento fakt uživatele upozorňuje a konverze neproběhne. Tímto je znemožněno zadat kontextovou či jinak nevyhovující gramatiku.

Definice SLL(k) gramatik platí pouze pro redukované bezkontextové gramatiky, proto je při volání metody *isSLL* nutné testovat vstupní gramatiku na redukovanost. Při této kontrole se gramatika načte z formuláře ve dvou kopiích, jedna z nich se zredukuje a poté se porovná s „originální“ gramatikou. Nejsou-li si rovny, není gramatika zapsaná ve formuláři redukovaná, o čemž je uživatel pomocí chybového hlášení informován a volaná metoda se neprovede. Uživatel může gramatiku zapsanou ve formuláři redukovat pomocí funkce, která se spustí stiskem tlačítka *Redukuj gramatiku*.

Je-li zadaná gramatika bezkontextová, lze nad ní počítat funkce FIRST(k) a FOLLOW(k) a v případě, že je také redukovaná, lze o ní rozhodovat, zda je SLL(k). Parametr *k* se zadává pro každou funkci zvlášť do příslušného vstupního pole. V případě vo-

lání funkce FIRST nebo FOLLOW musí ještě uživatel zadat parametr *Řetězec* respektive *Neterminál*, který bude argumentem volané funkce. Také formát parametrů je kontrolován tak, aby číslo *k* bylo celé kladné, vstupní řetězec obsahoval pouze symboly ze vstupní gramatiky a neterminál byl z *Množiny neterminálů*. Je-li některý z parametrů zadán nesprávně, je o tom uživatel informován odpovídajícím chybovým hlášením. Jsou-li parametry zadány správně, proběhne výpočet a jeho výsledek se zobrazí v textovém poli *Výsledky*.

4.3.3 Ostatní prvky

- Popisek *Chyba* – v této komponentě se zobrazují chybová hlášení. Tímto způsobem je uživatel informován, nastane-li nějaká chyba (obvykle, když je formulář špatně vyplněn).

- Textové pole *Výsledky* – do něho se vypisují výsledky volaných metod FIRST(k), FOLLOW(k) a SLL(k) a kontrolní hlášení některých dalších provedených funkcí.

- Tlačítka *Vymaž gramatiku* a *Vymaž výsledky* – po stisknutí těchto tlačítek dojde k nenávratnému vymazání obsahu všech čtyř stupních polí gramatiky (*Množiny neterminálů*, *Množiny terminálů*, *Počátečního neterminálu*, *Množiny pravidel*), respektive k vymazání obsahu textového pole *Výsledky*.

5 Uživatelský manuál

Následující kapitola popisuje možnosti a způsoby použití výsledné aplikace z pohledu uživatele.

5.1 Distribuce

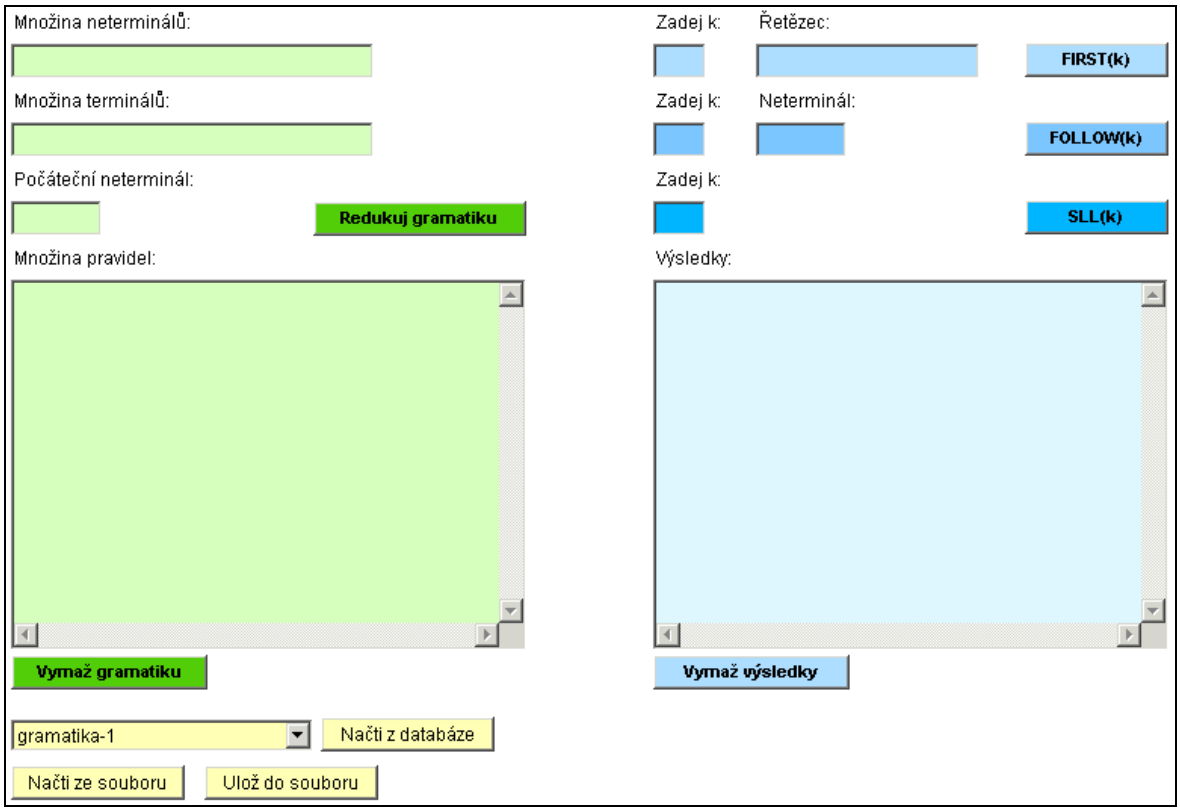
Program je distribuován jako *FormularApplet.jar* balíček, který obsahuje všechny vytvořené a přeložené třídy. Ten je použit ve vzorové HTML stránce *FormularApplet.html*, která je také součástí této práce.

5.2 Spuštění programu

Aplet se obvykle spouští jako součást HTML stránky. Pro zobrazení apletu stačí otevřít HTML soubor v prohlížeči webových stránek a mít jeho přeložený zdrojový kód ve stejném adresáři, jako se nachází tento HTML soubor. Je nutné podotknout, že prohlížeč, ve kterém chcete HTML stránku s apletem zobrazit, musí mít nainstalovánu stejnou nebo vyšší verzi jazyka Java, jako ve které byl aplet napsán (nejnovější verzi jazyka Java je možné si zdarma stáhnout na http://java.sun.com/j2se/downloads/). Další možností je použít program *Appletviewer*, který je součástí JDK. Tento program ale umí zobrazit pouze aplety, které jsou součástí HTML stránky uložené na lokálním disku a obvykle se používá jen pro ladění právě vytvářeného apletu.

Z bezpečnostních důvodů popsaných v části 4.3.1 *Vstupy a výstupy* se při spuštění apletu ve webovém prohlížeči zobrazí dialogové okno upozorňující na fakt, že spouštěný aplet je podepsán osobou „*Michal Hanzelka*“. Chcete-li, aby byl aplet zaveden a jeho funkce nebyly nijak omezeny, musíte zvolit možnost vyjadřující Váš souhlas se spuštěním apletu (obvykle tlačítko s nápisem „Yes“ nebo „Run“). Při jiné volbě (např. „*No*“ či „Cancel“), je sice aplet také spuštěn, ale funkce pracující se soubory na Vašem počítači nebudou fungovat.

Po spuštění apletu v prohlížeči se zobrazí webový formulář. Na jeho levé straně jsou pod sebou uspořádána vstupní pole pro zadání gramatiky a tlačítko na jejich vymazání, pod nimi se nacházejí tlačítka umožňující ukládat a načítat gramatiky do a ze souborů. V blízkosti vstupních polí gramatiky je umístěno tlačítko *Redukuj gramatiku*. V pravé části zobrazené stránky se nacházejí tlačítka pro výpočet funkcí a vstupní pole jejich parametrů. Vpravo dole je umístěno textové pole pro zobrazení výsledků volaných funkcí a pod ním tlačítko umožňující jeho vymazání.



Obrázek 5.1: Prázdný formulář

5.3 Ovládání programu

5.3.1 Ruční zadání gramatiky

Při ručním zadávání gramatik se jednotlivá pole vyplňují textem zadaným z klávesnice. Pro přepínání mezi kolonkami můžete použít klávesy *Tab*.

Do pole *Množina neterminálů* zapište jména neterminálních symbolů, do pole *Množina terminálů* jména terminálních symbolů. Pro volbu a zápis jmen neterminálů a terminálů je nutné, abyste dodrželi následující pravidla:

- Jména ve vstupních polích jsou od sebe oddělena právě jedním ze znaků ‘ ’(mezera), ‘,’ (čárka), ‘;’ (středník), ‘:’ (dvojtečka), ‘-’ (mínus), ‘>’ (větší než), ‘|’ (svislá čára). Pro všechny tyto znaky se dále v textu používá společný pojem *oddělovače*.

- Jména mohou obsahovat jakékoliv alfanumerické znaky a speciální znaky vyjma *oddělovačů*. V případě, že jméno symbolu bude obsahovat nějaký jiný znak, nemusí program fungovat správně.

- Jméno musí být v celé gramatice unikátní.

- Vícenásobný výskyt jednoho jména v množině není možný. Opakuje-li se některé jméno v poli *Množina neterminálů* nebo v poli *Množina terminálů* je jeho vícenásobný výskyt ignorován a program se chová, jakoby každé jméno bylo zadáno právě jednou.

- Pro všechna jména musí platit, že nejsou prefixem (předponou) žádného jiného jména gramatiky.

- Jako jméno epsilonu (prázdného slova) použijte řetězec *„eps“*.

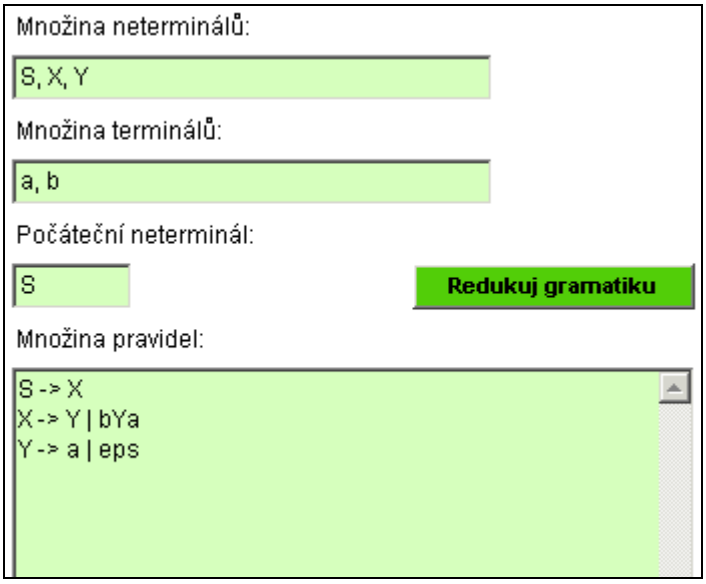
- Žádné jméno neterminálu nebo terminálu nesmí být prefixem jména vyhrazeného pro epsilon (*„eps“*).

Do pole *Počáteční neterminál* napište jméno počátečního (startovacího) neterminálu. Toto jméno musí být uvedeno v *Množině neterminálů*.

Pravidla gramatiky zadejte do textového pole *Množina pravidel*. V tomto poli uveďte na jednom řádku všechna pravidla pro jeden neterminál. První řádek by měl podle zvyklostí obsahovat pravidla pro počáteční neterminál. Každý řádek musí začínat právě jedním neterminálem, za kterým následuje minimálně jeden libovolný *oddělovač* a dále řetězce terminálních a neterminálních symbolů oddělených od sebe minimálně jedním *oddělovačem*. Zadáte-li více stejných pravidel pro jeden neterminál, program se chová, jakoby byla zadána pouze jednou. Každý neterminál může být uveden maximálně jednou na levé straně některého řádku v poli *Množina pravidel*.

Je-li alespoň jeden z výše popsaných požadavků porušen a stisknete-li některé z tlačítek *Redukuj gramatiku, FIRST(k), FOLLOW(k), SLL(k),* volaná funkce se neprovede a u horního okraje apletu se objeví chybové hlášení s informací, které pravidlo bylo porušeno.

Ukázka zápisu gramatiky splňující všechny kladené požadavky je na obrázku 5.2.



Obrázek 5.2 Zadání gramatiky

5.3.2 Načítání a ukládání gramatik

Aplikace umožňuje načítat gramatiky z externích souborů uložených na disku klienta a z databáze gramatik uložené na serveru. Gramatiku zapsanou ve formuláři je možno uložit do libovolného souboru na disk uživatele. Komponenty sloužící k načítání a ukládání gramatik se nacházejí v levé dolní části formuláři.

Pro načtení gramatiky z databáze uložené na serveru zvolte ve výběrovém menu (nachází se v levé dolní části formuláře) jméno gramatiky a poté stiskněte tlačítko *Načti z databáze*. Tím se vybraná gramatika vloží do formuláře.

Máte-li ve správném formátu zapsanou gramatiku v souboru (viz část 4.3.1 Vstupy a výstupy), můžete ji načíst do formuláře. Stiskněte tlačítko *Načti ze souboru*, v zobrazené adresářové struktuře vyhledejte soubor s gramatikou, kterou chcete nahrát a stiskněte tlačítko *Otevřít*. Obsah souboru se poté zobrazí ve formuláři.

Chcete-li uložit obsah formuláře do souboru na lokální disk, stiskněte tlačítko *Ulož do souboru*, zadejte jméno nového souboru (včetně přípony) nebo vyberte již existující soubor a stiskněte *Uložit*. Poté zkontrolujte, že v poli *Výsledky* přibyl text „*Gramatika byla uložena.*“

5.3.3 Redukce gramatik

Gramatiky, o nichž lze rozhodovat, zda jsou SLL(k), musí být *redukované*. Není-li gramatika zapsaná ve formuláři v redukovaném tvaru a stisknete tlačítko *SLL(k)*, zobrazí se chybové hlášení „*Chyba: zadaná gramatika není redukovaná.*“ a výpočet neproběhne.

Chcete-li redukovat vstupní gramatiku, stiskněte tlačítko *Redukuj gramatiku*. Tato funkce přečte gramatiku z formuláře, zredukuje ji a výslednou gramatiku opět zapíše zpět do formuláře. Pro kontrolu se do pole *Výsledky* připíše text „*Gramatika byla redukována.*“

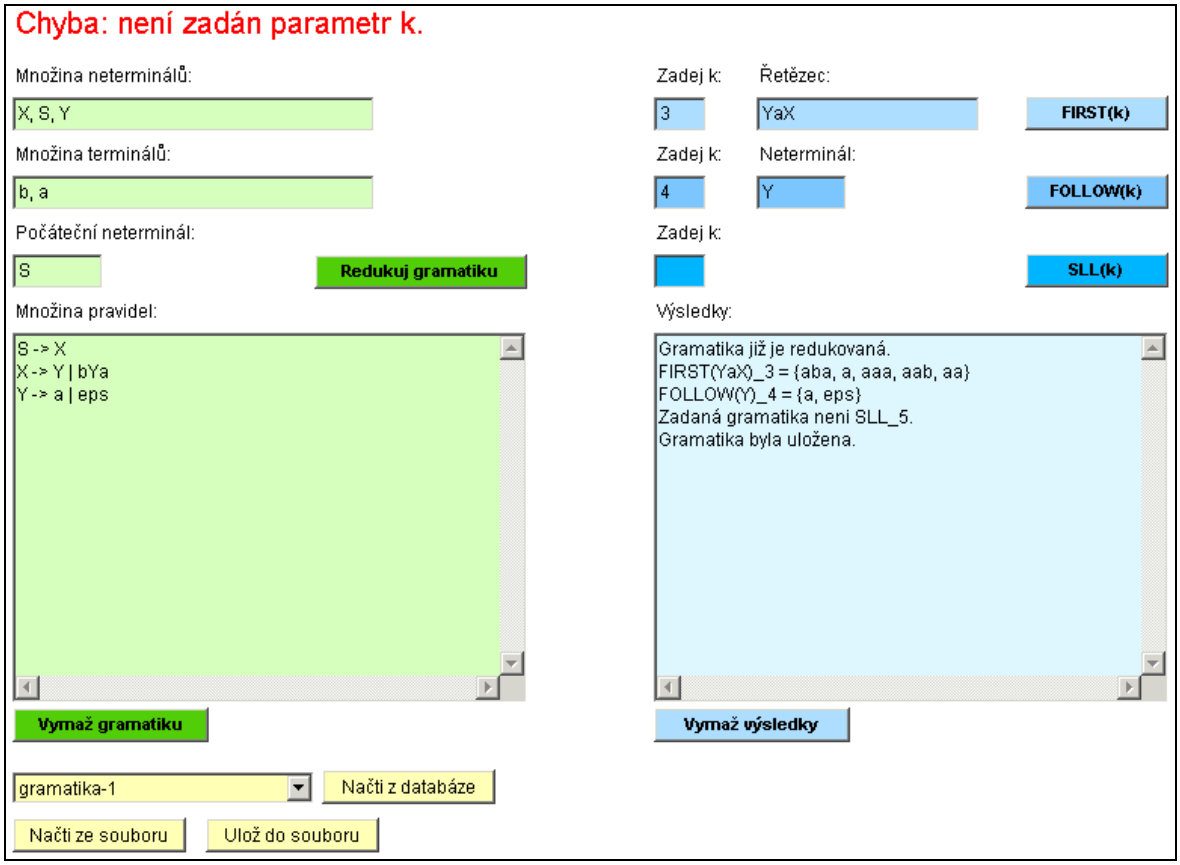
Jestliže je zadaná gramatika v redukovaném tvaru a stisknete tlačítko *Redukuj gramatiku*, proběhne pouze převedení obsahu vstupních polí do standardního tvaru, ve kterém jsou pravidla ve formátu „N -> α | β“ a na jejich prvním řádku jsou uvedena pravidla počátečního neterminálu.

5.3.4 Výpočet funkcí FIRST(k), FOLLOW(k) a SLL(k)

Je-li gramatika ve formuláři bezkontextová a je-li správně zapsána, lze nad ní počítat funkce FIRST a FOLLOW. Pro rozhodovaní zda se jedná o SLL(k) gramatiku, musí být navíc redukovaná.

Pro výpočet některé z funkcí nejdříve zadejte parametr *k* do kolonky nacházející ve stejném řádku jako tlačítko s názvem příslušné funkce. Parametrem *k* musí být celé kladné číslo. Chcete-li vypočítat funkci FIRST zadejte ještě vstupní řetězec skládající se pouze z terminálů a neterminálů zadané gramatiky. Pro výpočet funkce FOLLOW zadejte jeden neterminál z množiny neterminálů, pro který chcete tuto funkci vypočítat. Poté stiskněte tlačítko s názvem funkce, kterou chcete vypočítat a jsou-li gramatika i všechny parametry správně zadány, přidá se do pole *Výsledky* výsledek volané funkce. Doba výpočtu je závislá na volbě parametru *k,* složitosti vstupní gramatiky a na výkonu Vašeho počítače. Obecně pro *k* větší než deset může výpočet trvat i několik minut.

Na obrázku 5.3 je zobrazen stav formuláře po načtení gramatiky z databáze, stisknutí tlačítka *Redukuj gramatiku*, vyplnění parametrických polí, zavolání funkcí *FIRST(k)*, *FOLLOW(k)* a *SLL(k)*, uložení gramatiky do souboru a stisknutí tlačítka *SLL(k)* poté, co byl vymazán parametr *k* této funkce.



Obrázek 5.3: Ukázka použití formuláře

6 Závěr

Cílem této práce bylo vytvořit Java aplet, který by umožňoval počítat funkce FIRST a FOLLOW nad bezkontextovými gramatikami a rozhodoval o zadané gramatice, zda je SLL(k). Vytvořený Java aplet tyto funkce poskytuje se základní uživatelskou podporou.

Použité algoritmy nejsou nejefektivnější, jak z pohledu rychlosti, tak z pohledu paměťové náročnosti, ale zato jsou přehledné a lze z nich jednoduše vyčíst jejich principy. Také objektový návrh tříd reprezentujících gramatiku byl vytvořen tak, aby nebyl příliš složitý a tudíž „nečitelný“. Z tohoto důvodu jsem upustil od možnosti vytvořit pro každý typ symbolu (neterminál, terminál, epsilon) vlastní třídu, která by byla potomkem abstraktní třídy *Znak* a místo toho jsem do třídy *Znak* přidal atribut *typ*, který nese informaci o typu instance této třídy. Všechny třídy balíčku *dat_typ* jsem se snažil vytvořit tak, aby bylo možno využít tento balíček i v jiných programech, tedy aby nebyl závislý na implementaci vlastního grafického rozhraní.

Samotný formulář jsem se snažil vytvořit s přehledným a intuitivním ovládáním. To by se dalo ještě zjednodušit, například přednastavením některých hodnot nebo možností výběru parametru *k* z výběrového menu, ale domnívám se, že by tím utrpěla přehlednost a variabilita formuláře.

Použitá literatura

1. ČERNÁ, I., KŘETÍNSKÝ, M., KUČERA, A.: *Automaty a formální jazyky I*. Učební text, verze 1.3, Fakulta informatiky, Masarykova univerzita v Brně, 2003
2. ČERNÁ, I., BARNAT, J.: *Cvičení k předmětu IA006 Vybrané kapitoly z teorie automatů*. Fakulta informatiky, Masarykova univerzita v Brně, poslední modifikace 19. září 2003. Dostupné na Internetu: http://www.fi.muni.cz/~xbarnat/tafj/afj2cvika.ps (prosinec 2005)
3. HEROUT, P.: *Učebnice jazyka Java*. České Budějovice, Kopp, dotisk prvního vydání, 2000. ISBN 80-7232-115-3
4. HEROUT, P.: *Java – grafické uživatelské prostředí a čeština*. České Budějovice, Kopp, dotisk prvního vydání, 2004. ISBN 80-7232-237-0
5. HEROUT, P.: *Java – bohatství knihoven*. České Budějovice, Kopp, první vydání, 2003. ISBN 80-7232-209-5
6. *Java™ 2 Platform Standard Edition 5.0 API Specification* [online]. Dostupné na Internetu: http://java.sun.com/j2se/1.5.0/docs/api/ (prosinec 2005)

Příloha A

Obsahem přiloženého CD je text této práce ve formátech doc, pdf a ps. Dále všechny vytvořené zdrojové kódy, dokumentace a ukázková HTML stránka obsahující vytvořený aplet.