VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

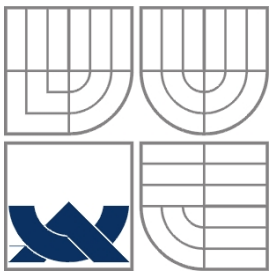
SYNTAKTICKÁ ANALÝZA ZALOŽENÁ NA STAVOVÝCH GRAMATIKÁCH

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

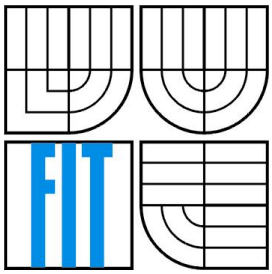
AUTOR PRÁCE
AUTHOR

MIROSLAV NOVOTNÝ

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYNTAKTICKÁ ANALÝZA ZALOŽENÁ NA STAVOVÝCH GRAMATIKÁCH

PARSING BASED ON STATE GRAMMARS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

Miroslav Novotný

VEDOUCÍ PRÁCE
SUPERVISOR

Meduna Alexander, prof. RNDr., CSc.

BRNO 2015

Abstrakt

Tato práce se zabývá syntaktickou analýzou založenou na stavových gramatikách. Cílem je vytvořit program schopný načíst gramatiku ze vstupního souboru. Na základě této gramatiky vytvořit LL tabulku a následně i provést syntaktickou analýzu zadaného vstupu. Na těchto základech pak studovat vlastnosti metod syntaktické analýzy, založené na těchto gramatikách. Testování probíhá i na gramatických strukturách, které nejsou bezkontextové.

Abstract

This thesis focuses on parsing based on state grammars. Its goal is to create a program, that will be able to load the grammar from input file. Based on a loaded grammar, the program will create an LL table and parse an input file using this table. The next goal is to study properties of parsing, based on state grammars, while using a created program as a stand point. Part of the testing will also be grammar structures which are not context-free.

Klíčová slova

Regulované gramatiky, stavové gramatiky, LL tabulka, syntaktická analýza

Keywords

Regulated grammars, state grammars, LL table, parsing

Citace

Novotný Miroslav: Syntaktická analýza založená na stavových gramatikách, bakalářská práce, Brno FIT VUT v Brně 2015

Syntaktická analýza založená na stavových gramatikách

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně, pod vedením pana profesora Alexandera Meduny.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Miroslav Novotný
28. 7. 2015

Poděkování

Tímto děkuji panu profesoru Medunovi za vedení a všechnu pomoc v rámci mé bakalářské práce.

Obsah

1 Úvod

V rámci informačních technologií se obor formálních jazyků zabývá jejich teorií a matematickou formalizací. Jazyk, jako takový, je velmi důležitým nástrojem předávání informací.

Přirozený jazyk, využívaný pro mezilidskou komunikaci, je značně složitý. Naopak pro ukládání úkolů procesoru je v závislosti na jeho architektuře využívána poměrně jednoduchá sekvence binárních hodnot. Ty jsou reprezentovány jazykem symbolických instrukcí, do kterého je překladačem vyššího programovacího jazyka převeden program napsaný programátorem. Vyšší programovací jazyky používají sice komplikovanější konstrukce než jazyk symbolických instrukcí, ale jejich složitost je stále značně nižší než složitost přirozeného jazyka. Přesto nám však programovací jazyky často umožňují například pracovat se značnou mírou abstrakce a strukturami jazyka popsat libovolný algoritmus. Tyto struktury mohou být složitější než v případě jazyka symbolických instrukcí, aby nám poskytly větší komfort při programování, ale jsou dostatečně jednoduché, abychom je mohli poměrně snadno popsat, formálně definovat, přeložit do jazyka symbolických instrukcí a využít je při tvorbě programu založeném na úkolu zadaném v přirozeném jazyce. K popisu složitějších jazyků je vhodné používat mocnější nástroje a jazykové aparáty. Programovací jazyky, potažmo jejich překladače a interprety, vychází z poznatků teorie formálních jazyků, což jasně dokládá jejich důležitost.

Záměrem této práce je prostudovat vlastnosti a chování stavových gramatik, jež jsou druhem regulovaných gramatik, jejich použití v různých situacích a také ve spolupráci s dalšími aparáty. Regulované gramatiky jsou založeny na klasických gramatikách, jsou tedy jejich rozšířením, které dokáže kontrolovat výběr pravidla použitého v rámci generování jazyka.

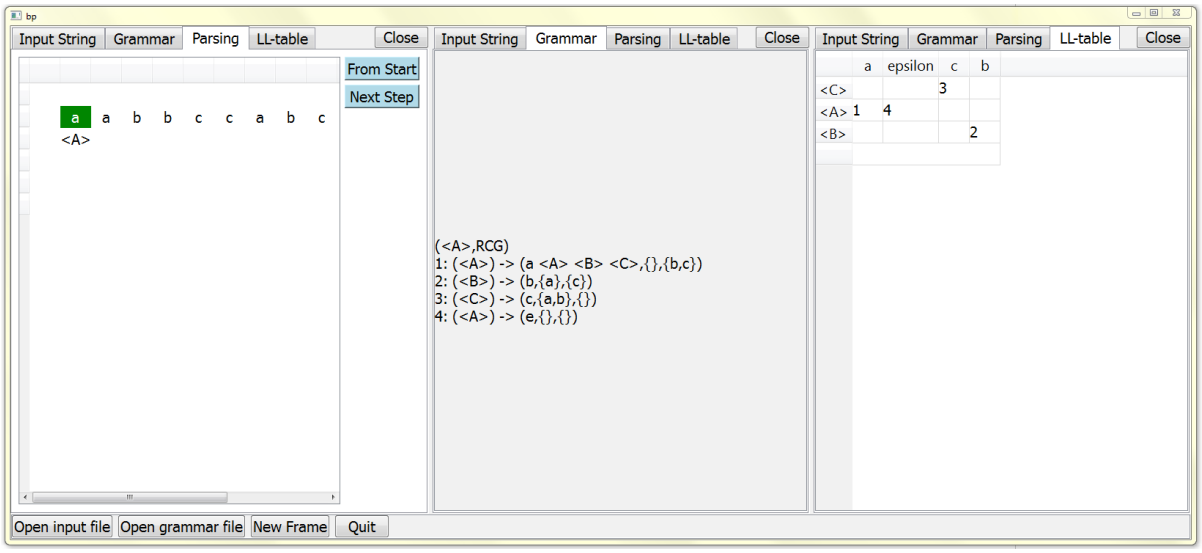
Cílem je také zjistit, zda a jak se hodí spojit stavové regulované gramatiky s analýzou založenou na LL tabulkách. Implementace začíná postupem LL(1) a postupně jsou přidávány další aparáty, jež mají za úkol zvýšit sílu gramatik a zároveň otestovat vhodnost využití tohoto spojení.

V rámci této aplikace je předpokládána kompletní determinističnost pravidel. Tento fakt se může ukázat jako omezující faktor co se týče síly gramatik, ale je nutný pro správnou funkci analýzy založené na nejlevější derivaci.

Jako rozšíření je uvažováno například úprava LL analýzy ze striktně nejlevějšího na nejlevější možný vstupní symbol, pro který existuje pravidlo gramatiky. Předpokladem je, že tato úprava zvýší množství přijatých jazyků beze změny na straně gramatiky. Riziko tohoto přístupu je ovšem ztráta kontroly, neboť nenabízí dostatek restriktivních prostředků pro pravidla. Z toho důvodu je zavedeno další rozšíření, které má zajistit zvýšení kontroly přidáním omezujících faktorů do jednotlivých pravidel. Tímto aparátem měly být gramatiky s nahodilým kontextem (*Random Context Grammars)*. Nakonec bylo nutné použít upravenou verzi těchto gramatik a proto definuji gramatiky s nahodilým terminálním kontextem (*Random Context Terminal Grammars*). Pomocí těchto úprav je možné silně zvýšit kontrolu nad generovaným jazykem zavedením restriktivních podmínek do pravidel v gramatice.

Jelikož součástí práce je i grafická reprezentace výše uvedených postupů, je za tímto účelem navrženo a implementováno za tímto účelem vhodné uživatelské rozhraní. Toto rozhraní obsahuje nástroje, které umožňují přehlednou prezentaci syntaktické analýzy a struktur k tomu použitých. Cílem také je, aby bylo rozhraní přehledné a snadno ovladatelné. Jedna z vlastností programu je též možnost přizpůsobit se změnám velikosti okna a změně množství zobrazovaných informací.

Následující kapitola této práce se věnuje základním použitým elementům z teorie formálních jazyků, jako jsou například jazyk a gramatika. Další část je věnována postupu analýzy při využití striktně nejlevější derivace. Tento postup je často používaný, a nejen proto velmi důležitý. Čtvrtá kapitola se zabývá teorií stavových gramatik, poněvadž je prostudování těchto gramatik jedním z hlavních cílů této práce, je nutné s ní čtenáře seznámit. Následuje seznámení se s aparátem nejlevější možné derivace, jež rozšiřuje postup syntaktické analýzy a tím i možnosti gramatik, co se týče přijatých řetězců. Následující kapitola se věnuje tématu gramatik s nahodilým kontextem. Tento přídavek má za účel rozšířit možnosti programu, jak co se týče typů gramatik, které dokáže převést na LL tabulky, tak rozsahu jazyků, které dokáže přijmout syntaktickým analyzátorem při použití těchto tabulek. Kapitola Grafická reprezentace pojednává o způsobu, jakým jsou informace předávány uživateli programu, a také o možnostech jeho ovládání. V kapitole věnující se implementaci jsou základní informace o struktuře programu a také je zde uvedeno v jakém jazyce a s jakými knihovnami byl vytvořen. V předposlední části práce je předvedeno základní použití programu a závěry, ke kterým práce míří. V závěru jsou vyjmenovány přínosy této práce a také jsou zde uvedeny možnosti případného rozšíření.



Obrázek 1.1: Ukázka z výsledného programu

2 Teorie a definice

Tato kapitola se zabývá výkladem pojmů problematiky formálních jazyků a jejich základy a některých dalších aparátů týkajících se této práce. Jsou zde uvedeny základní definice a je popsána LL analýza jazyka.

2.1 Základní definice

Kapitola čerpá z [4], pokud není uvedeno jinak.

Definice 2.1. *Abeceda* je konečná neprázdná množina elementů zvaných symboly.

Definice 2.2. *Slovo* nebo *řetězec* nad abecedou Σ je konečná posloupnost symbolů z Σ.

Definice 2.3. *Prázdný řetězec* je označován jako ε.

Definice 2.4. *Jazyk* L nad Σ je množina řetězců nad Σ. Množinu všech řetězců nad Σ nazýváme *univerzální jazyk*.

Definice 2.5. *Rodina jazyků* je dvojice nekonečné množiny symbolů Σ a množiny jazyků M. Pro každý jazyk L v této množině platí, že jeho abeceda je podmnožinou Σ. L nesmí být prázdná množina.

Definice 2.6. *Terminál* je jeden znak z řetězce z daného jazyka.

Definice 2.7. *Neterminál* je typ symbolu, který může být v rámci analýzy nebo generování přepsán dle vhodného pravidla.

Definice 2.8. *Nedeterministický konečný automat* [8] A je pětice (Q, T, δ, s, F), kde

1) Q je abeceda stavů
2) T je abeceda vstupních symbolů
3) $\delta: Q \times T \rightarrow 2^Q$ je přechodová funkce
   (2^Q označuje množinu všech podmnožin množiny Q)
4) $s \in Q$ je počáteční stav
5) $F \subseteq Q$ je množina koncových stavů.

Definice 2.9. Buď *V* abeceda. *Regulární výraz* [8] *R* nad *V* je definován rekurzivně

1) ∅ je regulární výraz označující regulární množinu ∅
2) ε je regulární výraz označující regulární množinu {ε}
3) jestliže $a \in V$, pak *a* je regulární výraz označující regulární množinu $\{a\}$
4) jestliže R_1 a R_2 jsou regulární výrazy označující regulární množiny L_1 a L_2, pak:
   i. $(R_1 + R_2)$ je regulární výraz označující regulární množinu $L_1 \cup L_2$
   ii. (R_1R_2) je výraz označující regulární množinu L_1L_2;
   iii. R_1^* je regulární výraz označující regulární množinu L_1^*
5) žádné jiné regulární výrazy, než vytvořené podle 1) až 4) nad abecedou *V* neexistují.

2.2 Gramatiky

Gramatika, nebo také formální gramatika, označuje soubor pravidel používaných k popisu jazyka. Jinými slovy, gramatika je množina pravidel, podle kterých lze sestavit řetězec daného jazyka. Dále gramatika popisuje pouze tvar řetězců, nikoliv význam.

Definice 2.10. *Gramatika* G je čtveřice G = (N,T,P,S), kde

1) N je abeceda neterminálů
2) T je abeceda terminálů, přičemž platí $N \cap T = \emptyset$;
3) $P \subseteq (N \cup T)^*N(N \cup T^*) \times (N \cup T)^*$ je konečná množina pravidel;
4) $S \in N$ a značí počáteční neterminál

Páry $(u,v) \in P$, nazýváme přepisovacími pravidly.

2.3 Chomského hierarchie

Chomského klasifikace jazyků dělí jazyky na čtyři skupiny. Dělení závisí na tvaru pravidel, jež obsahuje množina přepisovacích pravidel. Jednotlivé skupiny jsou označovány jako typ 0, typ 1, typ 2 a typ 3.

Definice 2.11. Gramatika typu 0 obsahuje pravidla v nejobecnějším tvaru, shodným s definicí gramatiky 2.9.:

$\alpha \rightarrow \beta$, $\alpha \in (N \cup \Sigma)^* N (N \cup \Sigma)^*$, $\beta \in (N \cup \Sigma)^*$.

Z tohoto důvodu se gramatiky typu 0 nazývají také gramatikami neomezenými.

Definice 2.12. Gramatika typu 1 obsahuje pravidla tvaru:

$\alpha A \beta \rightarrow \alpha \gamma \beta$, $A \in N$, $\alpha, \beta \in (N \cup \Sigma)^*$, $\gamma \in (N \cup \Sigma)^+$

nebo

$S \rightarrow \varepsilon$, pokud se S neobjevuje na pravé straně žádného pravidla.

Tyto gramatiky se také nazývají kontextové, neboť tvar jejich pravidel implikuje, že neterminál A může být nahrazen pouze tehdy, je-li jeho pravým kontextem řetězec β a levým kontextem řetězec α.

Definice 2.13. Gramatika typu 2 obsahuje pravidla tvaru:

$A \rightarrow \gamma$, $A \in N$, $\gamma \in (N \cup \Sigma)^*$

Těmto gramatikám se také říká bezkontextové.

Definice 2.14. Gramatika typu 3 obsahuje pravidla tvaru: $A \rightarrow xB$ nebo $A \rightarrow x$; $A, B \in N$, $x \in \Sigma^*$

Gramatiky s tímto tvarem pravidel se nazývají pravé lineární gramatiky. Jazyky generované těmito gramatikami lze přijímat konečným automatem a pro jakýkoliv z nich lze vytvořit ekvivalentní regulární výraz.

Definice převzaty z [9].

2.4 Bezkontextové gramatiky

Jedním z možných typů gramatik jsou *bezkontextové gramatiky*, tyto mají takovou vlastnost, že každé jejich pravidlo musí být zapsáno ve tvaru $V \rightarrow w$, kde V je neterminální symbol a w je řetězec terminálních a neterminálních symbolů. Tyto gramatiky při aplikaci pravidel, jak již název napovídá, neberou ohled na kontext v okolí neterminálu, na který je pravidlo aplikováno.

Jazyky, které jsou generovány bezkontextovými gramatikami, se nazývají bezkontextové jazyky. Bezkontextové gramatiky se rozšířily jako nástroj pro popis programovacích a značkovacích jazyků, pro něž jsou díky své jednoduchosti vhodné. Bezkontextová gramatika, je často jednoduchý způsob jak popsat nějakou gramatickou strukturu.

2.5 Regulované gramatiky

Regulovaná gramatika je bezkontextová gramatika rozšířená o další přídavný aparát, který určuje, jaká pravidla budou použita v rámci generování jazyka. Jedná se o velmi obecný pojem, a proto síla samotných gramatik z této skupiny závisí na typu použité gramatiky. Případem regulovaných gramatik jsou takzvané *stavové gramatiky*. Ty se vzájemně liší v tom, jak hluboko v zásobníku může být rozgenerovatelný neterminál (*n-limitované,* kde n je přirozené číslo ≥ 1).

Jednou z regulovaných gramatik je také gramatika s rozptýleným kontextem. Rozdíl této gramatiky od bezkontextové gramatiky je ten, že v každém kroku může být přepsáno více neterminálů než jen jeden. Tyto gramatiky se mohou také lišit, v závislosti na počtu v jednom kroku přepsaných neterminálů (n-limitované) [7].

Dalšími typy regulovaných gramatik jsou například *maticové gramatiky*. Tyto gramatiky přidávají do bezkontextové gramatiky množinu neprázdných řetězců. Těmto řetězcům se říká matice a jsou složeny z prvků množiny pravidel. Při výběru pravidla se uvažuje první pravidlo v matici. Po vybrání jsou následně aplikována všechna pravidla v pořadí, v němž se vyskytují. Po úspěšném ukončení derivace je možné ukončit celý proces, anebo vybrat další matici.

Síla regulovaných gramatik obvykle závisí na typu použité gramatiky, n-limitované gramatiky jsou také ovlivněny hodnotou čísla n. Minimální síla těchto gramatik odpovídá bezkontextovým jazykům.

2.6 Syntaktická analýza

Syntaktická analýza (*parsing)* je postup, při němž dochází k postupnému načítání vstupního řetězce nějakého jazyka. Tento řetězec bývá posloupností symbolů používaných analyzovaným jazykem. Cílem analýzy je zjistit, zda řetězec odpovídá pravidlům předem určené gramatiky. Výsledkem analýzy je obrácená posloupnost pravidel použitá k pravé derivaci řetězce, té se také říká *levý rozbor*. V rámci syntaktické analýzy často dochází k tvorbě *syntaktického stromu* nebo *derivačního stromu.*
Syntaktické analýze často předchází *lexikální analýza*, což je postup převedení vstupního řetězce znaků na řetězec lexikálních symbolů (*token*). Bývává implementován jako konečný automat.

2.7 LL-analýza

Celá podkapitola o LL-analýze čerpá z [8].
Existují bezkontextové jazyky, jež nelze přijmout žádným deterministickým (zásobníkovým) automatem. Existuje však také speciální typ bezkontextových gramatik, pro který lze vždy nalézt deterministický syntaktický analyzátor pracující shora dolů. Těmto gramatikám se říká LL gramatiky.
V rámci LL analýzy dochází k postupu zleva doprava a konstrukci nejlevější derivace řetězce. Tato analýza je založena na LL tabulce, která je zkonstruována pomocí LL gramatiky. Její struktura je uspořádána tak, aby pro každý jeden terminál a neterminál existovalo právě jedno pravidlo z klasické gramatiky. V případě stavových gramatik existuje právě jedno pravidlo pro jeden terminál, jeden neterminál a jeden stav současně.
Ne pro každou bezkontextovou nebo stavovou gramatiku lze vytvořit LL-tabulku. Existují snadno převoditelné gramatiky, těmto se říká jednoduché LL gramatiky.

Definice 2.15. Bezkontextová gramatika G = (N, T, P, S) se nazývá *jednoduchá LL(1) gramatika*, jestliže splňuje tyto dvě podmínky:

1. Pravá strana libovolného pravidla začíná terminálem
2. Jestliže dvě pravidla mají stejnou levou stranu, pak se liší v terminálu, kterým začíná jejich pravá strana.

Příklad 2.1. Jednoduchá LL(1) gramatika

1: <S> → a <A>
2: <A> → b <B>
3: <A> → c <S>
4: <B> → c <S>
5: <S> → ε

Příklad 2.2. LL(1) tabulka, dle gramatiky z příkladu **2.1.**

	a	b	c	ε
<S>	1			5
<A>		2	3	
<B>			4	

Na rozborovou tabulku jde však převést i složitější typ gramatik.

Pro G = (N,T,P,S), bezkontextová gramatika
Definice 2.16. FIRST(x), kde $x \in (N \cup T)^*$

$$FIRST(x) = \{a: x \Rightarrow^* ay, a \in T, y \in (N \cup T)^* \} \cup \{\varepsilon: x \Rightarrow^* \varepsilon \}$$

tedy, FIRST(x) je množina všech neterminálních symbolů, které se vyskytují na počátku řetězců odvoditelných z x.

Definice 2.17. FOLLOW(A), kde $A \in N$

$$FOLLOW(A) = \{c: S \Rightarrow^* xAy, y \Rightarrow^* cz, c \in T, x, y,z \in (N \cup T)$$

to znamená, že FOLLOW(A) je množina všech terminálních symbolů, které mohou být ve větných formách bezprostředně za symbolem A. Pokud existuje větná forma, jejíž posledním symbolem je A, pak množina FOLLOW(A) obsahuje rovněž ε.

Definice 2.18.

Bezkontextová gramatika G = (N, T, P, S) se nazývá LL(1) gramatika, právě když pro každý neterminál $A \in N$ platí:

Jestliže $A \rightarrow x$, $A \rightarrow y \in P$ a $x \neq y$ pak

$$FIRST(x\ FOLLOW(A)) \cap FIRST(y\ FOLLOW(A)) = \varnothing$$

Pro každou LL(1) gramatiku platí, že se z ní dá vytvořit LL tabulka.

Množinu FIRST vytvoříme následujícím způsobem.

V počátku platí, že $FIRST(x) = \varnothing$. Tato pravidla potom aplikujeme do té doby, dokud je možné nějakou množinu FIRST rozšířit o nový terminál nebo o ε.

1. Jestliže $x \in T$, pak $FIRST(x) = \{x\}$
2. Pokud $x \in N$ a $x \rightarrow \varepsilon \in P$ nebo $x = \varepsilon$, pak $FIRST(x) = FIRST(x) \cup \{\varepsilon\}$
3. Když $x \in N$ a $x \rightarrow ay$, $a \in T$, $y \in (N \cup T)^*$ nebo $x = ay$, pak $FIRST(x) = FIRST(x) \cup \{a\}$
4. Jestliže $X \in N$ a $X \rightarrow A1A2\ldots..Ak \in P$, $Ai \in N \cup T$, $1 \leq f \leq k$, $k \geq 1$

pak pro všechna n ($\leq$ k) taková, že $A1,\ldots., An\text{-}1 \in N$ a $e \in FIRST(Aj)$ pro $1 \leq j \leq n\text{-}1$ položíme $FIRST(X) = FIRST(X) \cup (FIRST(An) - \{e\})$; jestliže $e \in FIRST(Aj)$ pro každé $j = 1, \ldots, k$, pak $FIRST(X) = FIRST(X) \cup \{e\}$

Konstrukce množiny FOLLOW, probíhá následujícím způsobem.

V počátku máme FOLLOW(x) = ∅.

1. Jestliže X = S , pak přidej e do FOLLOW(X).
2. Jestliže $A \rightarrow uXv \in P$, $A \in N$, u, v, $\in (N \cup T)^*$, pak do FOLLOW(X) přidáme všechny terminály z FIRST(v)
3. Jestliže $A \rightarrow uX \in P$ nebo $A \rightarrow uXv \in P$ kde $e \in$ FIRST(v), $A \in N$, v, $u \in (N \cup T)^*$, pak do FOLLOW(X) přidej FOLLOW(A).

Tímto máme vytvořené obě množiny, FIRST a FOLLOW, přecházíme k následujícímu postupu:

Symbol # na vrcholu zásobníku značí prázdný vstupní řetězec.

1. Jestliže $A \rightarrow x$ je i-té pravidlo z P a $a \in$ FIRST(x) – {e}, pak M(A, a) = x, i
2. Jestliže $A \rightarrow x$ je i-té pravidlo z P, $e \in$ FIRST(x) a $b \in$ FOLLOW(A), pak M(A, b) = x, i
3. Jestliže $a \in T$, pak M(a, a,) = X;
4. M(#, e) = +
5. Ve všech zbývajících případech je na příslušném průsečíku sloupce a řádku – (tj. signalizace chyby).

Samotná analýza probíhá tak, že se postupně aplikují pravidla. K výběru pravidla dochází dle vrchní položky v zásobníku, v níž se na začátku nachází počáteční neterminál, a nejlevějšího vstupního symbolu (*tokenu*). Při aplikaci pravidla dochází k rozgenerování jeho pravé strany v opačném pořadí na vrchol zásobníku. Tímto způsobem se pokračuje, dokud není zásobník prázdný. Pokud dojde k vyprázdnění zásobníku, ale vstupní řetězec není ještě zpracován celý, pak řetězec není syntaktickým analyzátorem přijat. Stejně tak pokud již dojde vstup, ale zásobník ještě není prázdný. Řetězec je úspěšně přijat pouze tehdy, dojde-li současně k vyprázdnění zásobníku a dočtení celého vstupního řetězce.
LL-analýza se může dále lišit v závislosti na počtu vstupních symbolů, které jsou ze vstupu brány při výběru pravidla. Pak LL(1) vezme ze vstupního řetězce jeden symbol a použije jej pro nalezení pravidla, zatímco LL(2) vezme dva symboly a pak hledá pravidlo, které začíná právě těmito dvěma symboly.
Existuje i LL(0), ta vychází z gramatiky, jež má pro každý neterminál pouze jedno pravidlo, nedochází k žádnému hledání pravidla na základě vstupu, ten se pouze čte. Tento přístup je pro syntaktickou analýzu nepoužitelný.

2.8 Striktně nejlevější derivace

Striktně nejlevější derivace bývá také nazývána nejlevější derivace typu 1. V rámci každého derivačního kroku dochází k přepsání nejlevějšího výskytu neterminálu.
Tento přístup k syntaktické analýze, kdy za pomoci LL tabulky probíhají derivace v přesně daném pořadí, umožňuje jednoduchou implementaci a transparentní průběh analýzy. Tento postup je často používán při tvorbě programovacích jazyků.
Tvar LL tabulky přímo závisí na daných pravidlech, k jejímu vytvoření tedy stačí pouze samotná pravidla a algoritmus pro převedení založený na množinových operacích.
V rámci práce budeme uvažovat tento tvar pravidel:
[číslo pravidla]: [levý neterminál] → [sekvence terminálů a neterminálů]
například 1: <A> → a b c <A>

Jako příklad použijeme jazyk $a^n b^n$.
Klasická gramatika se může skládat například z pravidel:
1: <S> → a <S> b
2: <S> → ε

Kde <S> je počáteční neterminál.

Pro tuto gramatiku existuje LL tabulka:

	a	ε
<S>	1	2

Pak se levý rozbor řetězce aaaabbbb sestává z posloupnosti pravidel 1, 1, 1, 1, 2.

Pozn. **Tučně** je označen nejlevější symbol na vstupu. Ve druhém sloupci se střídá vstupní řetězec se zásobníkem.

Pro řetězec aabb by probíhala analýza takto:

Krok 1.	**a** a b b	Začátek syntaktické analýzy.
	<S>	Původní stav.
Krok 2.	**a** a b b	Rozgenerování neterminálu <S>.
	a <S> b	Použití pravidla 1.
Krok 3.	**a** b b	Přijetí terminálu 'a'.
	<S> b	
Krok 4.	**a** b b	Rozgenerování neterminálu <S>.
	a <S> b b	Použití pravidla 1.
Krok 5.	**b** b	Přijetí terminálu 'a'.
	<S> b b	Rozgenerování neterminálu <S>.
Krok 6.	**b** b	Použití pravidla 2.
	b b	
Krok 7.	**b**	Dvě následná přijetí terminálu 'b'.
	b	

Syntaktická analýza proběhla úspěšně, řetězec patří do jazyka generovaného výše zmíněnou gramatikou.

1: <A> → a <B>
2: <B> → b <C>
3: <C> → c <A>
4: <A> → ε

Této gramatice odpovídá tabulka:

	a	b	c	ε
<A>	1			4
<B>		2		
<C>			3	

Analýza abcabc pak probíhá za použití pravidel 1, 2, 3, 1, 2, 3, 4.

Pro řetězec abc by probíhala analýza takto:

Krok 1.	**a** b c	Začátek syntaktické analýzy.
	<A>	Původní stav.
Krok 2.	**a** b c	Rozgenerování neterminálu <A>.
	a <B>	Použití pravidla 1.
Krok 3.	**b** c	Přijetí terminálu 'a'.
	<B>	
Krok 4.	**b** c	Rozgenerování neterminálu <B>.
	b <C>	Použití pravidla 2.
Krok 5.	**c**	Přijetí terminálu 'b'.
	<C>	
Krok 6.	**c**	Rozgenerování neterminálu <C>.
	c <A>	Použití pravidla 3.
Krok 7.	ε	Přijetí terminálu 'c'.
	<A>	
Krok 8.		Rozgenerování neterminálu <A>.
		Použití pravidla 4.

Syntaktická analýza proběhla úspěšně, řetězec patří do jazyka generovaného výše zmíněnou gramatikou.

Obsah této podkapitoly je úzce spjatý se samotnou praktickou částí bakalářské práce, jelikož tvoří jádro syntaktické analýzy implementace. Požadovaných cílů je dosaženo rozšiřováním tohoto přístupu.

2.9 Stavové gramatiky

V této podkapitole se seznámíme se stavovými gramatikami, jejich silou a výsledkem jejich kombinace s LL analýzou.

Regulované gramatiky jsou obecně snahou o zvýšení množství jazyků generovaných danými gramatikami, pomocí přidání dalšího řídícího aparátu, který by umožňoval ovlivňovat, jaké pravidlo bude použito a kdy. V případě stavových gramatik, je do gramatik přidán systém, připomínající konečný automat. V rámci každého provedení nějakého pravidla, dojde kromě přepisu neterminálu i ke změně stavu a tedy k ovlivnění výběru dalšího pravidla.
Pokud k aplikaci pravidla vždy dojde v rámci prvních n neterminálů na zásobníku, gramatika je považována za n-limitovanou.

Definice 2.19. *Stavová gramatika* je pětice G = (V, W, T, P, S), kde

1) V je celková abeceda
2) W je konečná množina všech stavů
3) T je abeceda terminálů, přičemž $T \subseteq V$
4) S je počáteční symbol, $S \in V - T$
5) $P \subseteq (W \times (V\text{-}T)) \times (W \times V^{+})$

Generativní síla těchto gramatik silně závisí na jejich n-limitovanosti. Pomocí ST označíme rodinu jazyků generovanou stavovými gramatikami. Pro každé $n \geq 1$, ST_n značí rodinu jazyků generovaných n-limitovanou stavovou gramatikou. ST_∞ značíme sjednocení všech rodin jazyků pro n od 1 do ∞.

Pak generativní síla ST_1 odpovídá bezkontextovým jazykům a ST_∞ odpovídá jazykům kontextovým.

Převzaté z [3].

Kombinace LL(1) analýzy se stavovými gramatikami pak odpovídá ST1, poněvadž tento typ analýzy pracuje pouze s vrcholem zásobníku.

V rámci práce a programu s ní spjatém pracuji s pravidly v tomto tvaru:
[číslo pravidla]: ([výchozí stav],[levý neterminál]) → ([stav po aplikaci pravidla],[sekvence terminálů a neterminálů])

například 1: (start,<A>) → (end,a b c <A>)

Jako příklad použijeme jazyk $a^n b a^n$.
Gramatika za použití stavových pravidel:

1: (1,<A>) → (1,a <A> <A>)
2: (1,<A>) → (2,b)
3: (2,<A>) → (2,a)

Kde <A> je počáteční neterminál.

Stav 1	a	b	Stav 2	a
<A>	1	2	<A>	3

Pro řetězec a a a a b a a a a budou použita pravidla 1, 1, 1, 1, 2, 3, 3, 3, 3.

Pozn. Ve druhém sloupci je před svislou čárou označen stav, ve kterém se automat v daném kroku nachází.
Pro řetězec aba by probíhala analýza takto:

Krok 1.	**a** b a	Začátek syntaktické analýzy.
	1 | <A>	Původní stav.
Krok 2.	**a** b a	Rozgenerování neterminálu <A>.
	1 | a <A> <A>	Použití pravidla 1.
Krok 3.	**b** a	Přijetí terminálu 'a'.
	1 | <A> <A>	
Krok 4.	**b** a	Rozgenerování neterminálu <A>.
	2 | b <A>	Použití pravidla 2.
Krok 5.	**a**	Přijetí terminálu 'b'.
	2 | <A>	
Krok 6.	**a**	Rozgenerování neterminálu <A>.
	2 | a	Použití pravidla 3.
Krok 7.		Přijetí terminálu 'a'.
	2 |	

3 Nejlevější možná derivace

V této kapitole rozšíříme možnosti analýzy o nový aparát. Namísto odebrání nejlevějšího symbolu na vstupu k nalezení pravidla použijeme nejlevější možný symbol, pro který existuje pravidlo. Metoda načtení prvního vhodného symbolu ze vstupního řetězce byla použita po konzultaci s vedoucím práce. Jedná se o úpravu metody, ve které se místo striktně nejlevějšího rozgeneruje nejlevější možný neterminál na zásobníku automatu.

3.1 Použití

S tímto aparátem dochází při výběru pravidla k procházení vstupního řetězce zleva, dokud není nalezen symbol, pro který existuje pravidlo. V případě, že algoritmus dojde až na konec řetězce, je možné ještě použít pravidlo s řetězcem nulové délky neboli ε.
Jelikož se rozdíl nachází v algoritmu syntaktické analýzy, nedochází k žádné změně tvaru načítaných pravidel nebo jejich převedení na LL tabulku. Pokud dochází v některém z následných kroků k přijetí symbolu, musí být přijat právě ten symbol, který byl použit při výběru pravidla.

3.2 Příklad s $a^nb^nc^n$

Jazyk $a^nb^nc^n$ bývá často používán jako příklad jazyka, který není bezkontextový, ale kontextový. Klasické gramatiky nebo i stavové gramatiky za použití LL(1) analýzy na něj tedy nestačí. V případě rozšíření LL(1) o hledání nejlevější možné derivace je možné napsat gramatiku, která tento jazyk generuje.

Tato gramatika se sestává například z pravidel:
1: <A> → a <B>
2: <B> → b <C>
3: <C> → c <A>
4: <A> → ε
Kde <A> je počáteční neterminál.
Těmto pravidlům odpovídá LL(1) tabulka:

	a	b	c	ε
<A>	1			4
<B>		2		
<C>			3	

Pak budou pro řetězec a a a a a b b b b b c c c c c použita pravidla:
1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 4

Pro řetězec aabbcc by probíhala analýza takto:

Pozn.: *Kurzívou* jsou označeny symboly přijaté z nejlevější možné pozice. **Tučně** jako v předchozích příkladech nejlevější symbol, pokud může být použit.

Krok 1.	**a** a b b c c	Začátek syntaktické analýzy.
	<A>	Původní stav.
Krok 2.	**a** a b b c c	Rozgenerování neterminálu <A>.
	a <B>	Použití pravidla 1.
Krok 3.	a *b* b c c	Přijetí terminálu 'a'.
	<B>	
Krok 4.	a *b* b c c	Rozgenerování neterminálu <B>.
	b <C>	Použití pravidla 2.
Krok 5.	a b *c* c	Přijetí terminálu 'b'.
	<C>	
Krok 6.	a b *c* c	Rozgenerování neterminálu <C>.
	c <A>	Použití pravidla 3.
Krok 7.	**a** b c	Přijetí terminálu 'c'.
	<A>	
Krok 8.	**a** b c	Rozgenerování neterminálu <A>.
	a <B>	Použití pravidla 1.
Krok 9.	**b** c	Přijetí terminálu 'a'.
	<B>	
Krok 10.	**b** c	Rozgenerování neterminálu <B>.
	b <C>	Použití pravidla 2.
Krok 11.	**c**	Přijetí terminálu 'b'.
	<C>	
Krok 12.	**c**	Rozgenerování neterminálu <C>.
	c <A>	Použití pravidla 3.
Krok 13.		Přijetí terminálu 'c'.
	<A>	
Krok 14.		Rozgenerování neterminálu <A>.

Zásobník i vstupní řetězec jsou prázdné, řetězec patří do jazyka generovaného touto gramatikou.

Pro srovnání: při použití přístupu s LL(1) prvního typu tento analyzátor přijímá $(abc)^n$, jak je uvedeno na příkladu u LL(1) gramatik.

3.3 Problémy tohoto přístupu

S pomocí tohoto typu LL analýzy můžeme přijímat jazyk $a^n b^n c^n$. Tento přístup nám ovšem neumožňuje dostatek restriktivní moci k tomu, abychom přijímali gramatikou pouze tento jazyk. Gramatika uvedená v předchozím příkladu sice dovede přijmout $a^n b^n c^n$, přijímá ovšem i další řetězce. Abychom byli konkrétní, tato gramatika akceptuje všechny řetězce, ve kterých se vyskytuje stejný počet symbolů 'a', 'b' a 'c'. Jazyk $a^n b^n c^n$ je tedy podmnožinou jazyka generovaného touto gramatikou.

Uvažujeme stejnou gramatiku, jako v předchozím příkladu. Tentokrát bude vstupní řetězec cba.

Krok 1.	c b a	Začátek syntaktické analýzy.
	<A>	Původní stav.
Krok 2.	c b *a*	Rozgenerování neterminálu <A>.
	a <B>	Použití pravidla 1.
Krok 3.	c *b*	Přijetí terminálu 'a'.
	<B>	
Krok 4.	c *b*	Rozgenerování neterminálu <B>.
	b <C>	Použití pravidla 2.
Krok 5.	**c**	Přijetí terminálu 'a'.
	<C>	
Krok 6.	**c**	Rozgenerování neterminálu <C>.
	c <A>	Použití pravidla 3.
Krok 7.		Přijetí terminálu 'c'.
	<A>	
Krok 8.		Rozgenerování neterminálu <A>.

V tomto příkladu je několikrát použito výběru terminálu pomocí analýzy druhého typu.

4 Další řídicí aparát

V této kapitole jsou uvedeny možnosti použití nahodilého kontextu v gramatikách, jako například Random Context Grammars, pro zvýšení restriktivní síly v jednotlivých pravidlech. Na základě tohoto typu gramatiky definuji Random Context Terminal Grammars (*Gramatiky s nahodilým terminálním kontextem*).
Informace o Random Context Grammars čerpány z [6].

4.1 Definice

V rámci tohoto aparátu dochází k rozšíření, ve kterém je ke každému pravidlu přidána množina zakazujících symbolů, tyto symboly se nesmí nacházet před, respektive za, načítaným symbolem.

Definice 4.1: Gramatika s nahodilým terminálním kontextem je pětice G = (N, T, P, S), kde

1) N je abeceda neterminálů
2) T je abeceda terminálů
3) P je konečná množina pravidel ve tvaru
   [A → x, U, W]
   kde A ∈ N, x ∈ (N ∪ T)*, a U,W ⊆ T;
4) S je počáteční neterminál

Tato gramatika při výběru pravidla bere v potaz všechny terminální symboly nalevo a napravo od symbolu, pro který je aktuálně hledáno pravidlo. Pokud bude pravidlo vyžadovat, aby se některý symbol vyskytoval pouze nalevo, respektive napravo, od aktuálního symbolu, nebude možné pravidlo použít.

4.2 Použití

Tento přístup umožňuje přesnější omezení množiny generovaných jazyků. Řeší tedy problém, na který je popsán v minulé kapitole. Zatímco s analýzou s nejlevějším možným akceptovatelným symbolem dokážeme přijmout $a^nb^nc^n$, ale nedokážeme přijmout *pouze* $a^nb^nc^n$.
Pravidlo obsahuje dvě množiny zakazujících terminálů. První množina určuje, které terminály se musí vyskytovat před místem použití pravidla a nesmí se vyskytovat za ním. Druhá množina naopak přikazuje, jaké terminály se musí vyskytovat za místem použitím pravidla, a které se nesmí vyskytovat před ním.

Použitý tvar pravidel je:
[číslo pravidla]: ([levý neterminál]) → ([sekvence terminálů a neterminálů],[{množina terminálů}],[{množina terminálů}])
Příklad:
2:(<A>) → (b, {a},{c})

nebo v případě stavové gramatiky:
[číslo pravidla]: ([výchozí stav],[levý neterminál]) → ([stav po aplikaci pravidla],[sekvence terminálů a neterminálů],[{množina terminálů}],[{množina terminálů}])
Příklad:
2:(p0,<A>) → (p1,b, {a},{c})

4.3 Příklad s $a^n b^n c^n$

Test bude probíhat s řetězcem aabbccabc, který neodpovídá jazyku $a^n b^n c^n$. Tento řetězec by však byl bez použití gramatik s nahodilým kontextem přijat.
Testovací gramatika:
(<A>,RCG)
1:(<A>) → (a <A> <B> <C>, {},{b,c})
2:(<A>) → (b, {a},{c})
3:(<A>) → (c, {a,b},{})
4:(<A>) → (ε, {},{})

Syntaktická analýza se zastaví s neúspěchem po použití pravidel v tomto to pořadí: 1, 1, 4, 2, 3, 2, 3. Zásobník je již prázdný a na vstupu stále zůstávají znaky 'a', 'b' a 'c'.

Další test stejné gramatiky, tentokrát na řetězec aaaaabbbbbccccc, který odpovídá $a^n b^n c^n$.

Postup výpisu programu je dále následující:

Krok 1.	
a a a a a b b b b b c c c c c	Původní vstupní řetězec.
<A>	Počáteční neterminál.
Krok 2.	
a a a a a b b b b b c c c c c	
a <A> <B> <C>	Použití pravidla 1.
Krok 3.	
a a a a b b b b b c c c c c	
<A> <B> <C>	
Krok 4.	
a a a a b b b b b c c c c c	
a <A> <B> <C> <B> <C>	Použití pravidla 1.
Krok 5.	
a a a b b b b b c c c c c	
<A> <B> <C> <B> <C>	

V krocích 6 až 10 analýza dále pokračuje obdobným způsobem. Dochází k opakované aplikaci pravidla 1, dokud nedojde k přijetí všech terminálních symbolů a.

Krok 11.
b b b b b c c c c c
<A> <B> <C> <B> <C> <B> <C> <B> <C> <B> <C>
Krok 12.
b b b b b c c c c c
ε <B> <C> <B> <C> <B> <C> <B> <C> <B> <C>
Krok 13.
b b b b b c c c c c
<B> <C> <B> <C> <B> <C> <B> <C> <B> <C>
Krok 14.
b b b b b c c c c c
b <C> <B> <C> <B> <C> <B> <C> <B> <C>
Krok 15.
b b b b c c c c c
<C> <B> <C> <B> <C> <B> <C> <B> <C>
Krok 16.
b b b b **c** c c c c
c <B> <C> <B> <C> <B> <C> <B> <C>
Krok 17.
b b b b c c c c
<B> <C> <B> <C> <B> <C> <B> <C>
…
Krok 33.

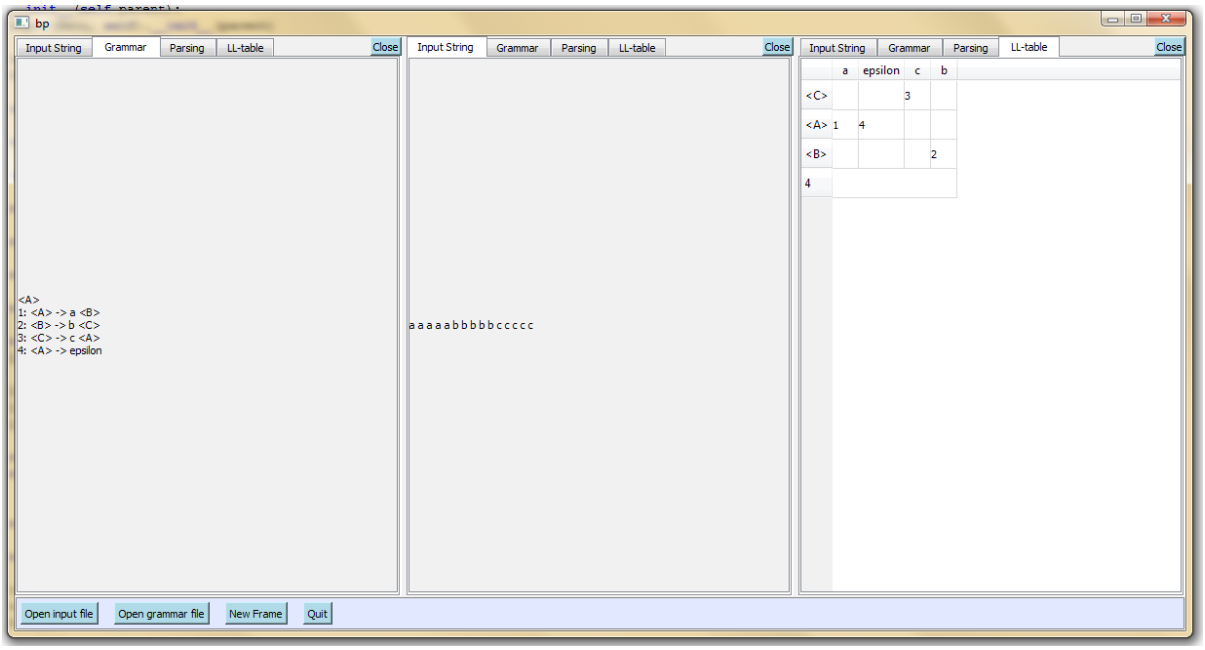
Poznámka ke kroku 11: došlo k aplikaci pravidla 4, protože již ani pro nejlevější možný symbol, který je součástí řetězce, nebylo nalezeno pravidlo (automat v rámci dopředného průchodu došel na konec řetězce), začne se jako vstup generovat ε. Pro neterminál <A> a ε již existuje pravidlo, a to již zmíněné pravidlo 4.

Syntaktická analýza byla úspěšná, použitá pravidla: 1, 1, 1, 1, 1, 4, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3

5 Grafická reprezentace

Součástí bakalářské práce je také grafická reprezentace. Cílem bylo vytvořit uživatelské rozhraní umožňující výběr vstupních souborů, prohlédnutí syntaktických struktur a přehledné krokování průběhu syntaktické analýzy.

5.1 Celkový pohled



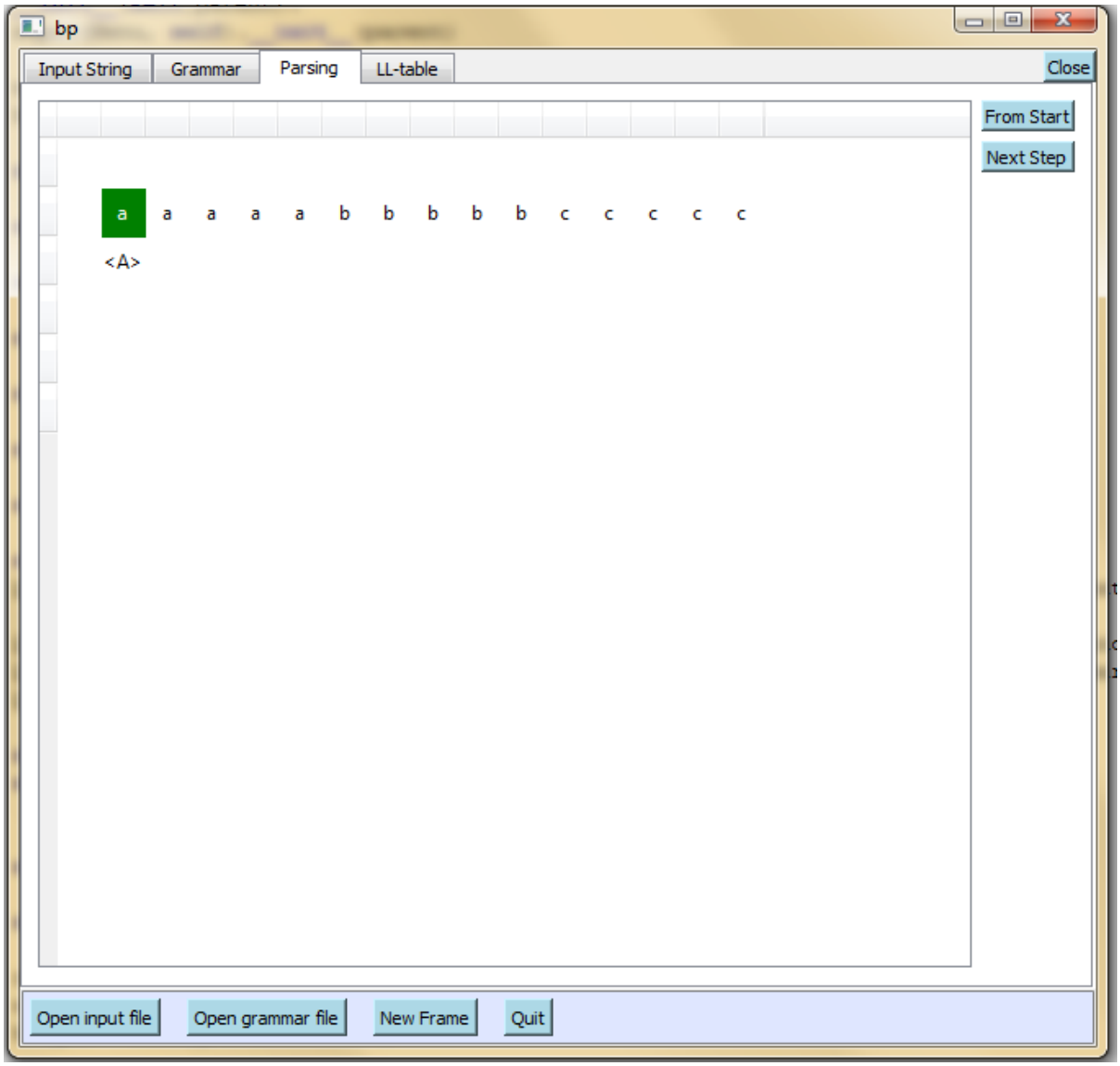
Obrázek 5.1: Příklad programu s několika informačními panely

Zde je příklad, jak vypadá program za běhu, v následujících odstavcích budou popsána jednotlivá okna, nejprve však spodní panel. Na řádku dole pod hlavními okny programu se nacházejí tři tlačítka. První dvě tlačítka, `Open input file` a `Open grammar file`, otevřou nové okno, které umožní načíst nový vstupní soubor, řetězec jazyka nebo gramatiku. Následující tlačítko, `New Frame`, vloží do hlavního okna nový panel, který v sobě bude obsahovat nejnovější načtenou gramatiku a vstupní řetězec a bude se na něm dát zvolit jeho obsah. Poslední tlačítko, `Quit`, ukončí program.

V rámci každého panelu nebo-li rámu jsou čtyři možnosti měnící typ obsahu, který je v něm zobrazený. Jsou to části `Input String`, `Grammar`, `Parsing` a `LL-table`.

5.2 Panel Parsing

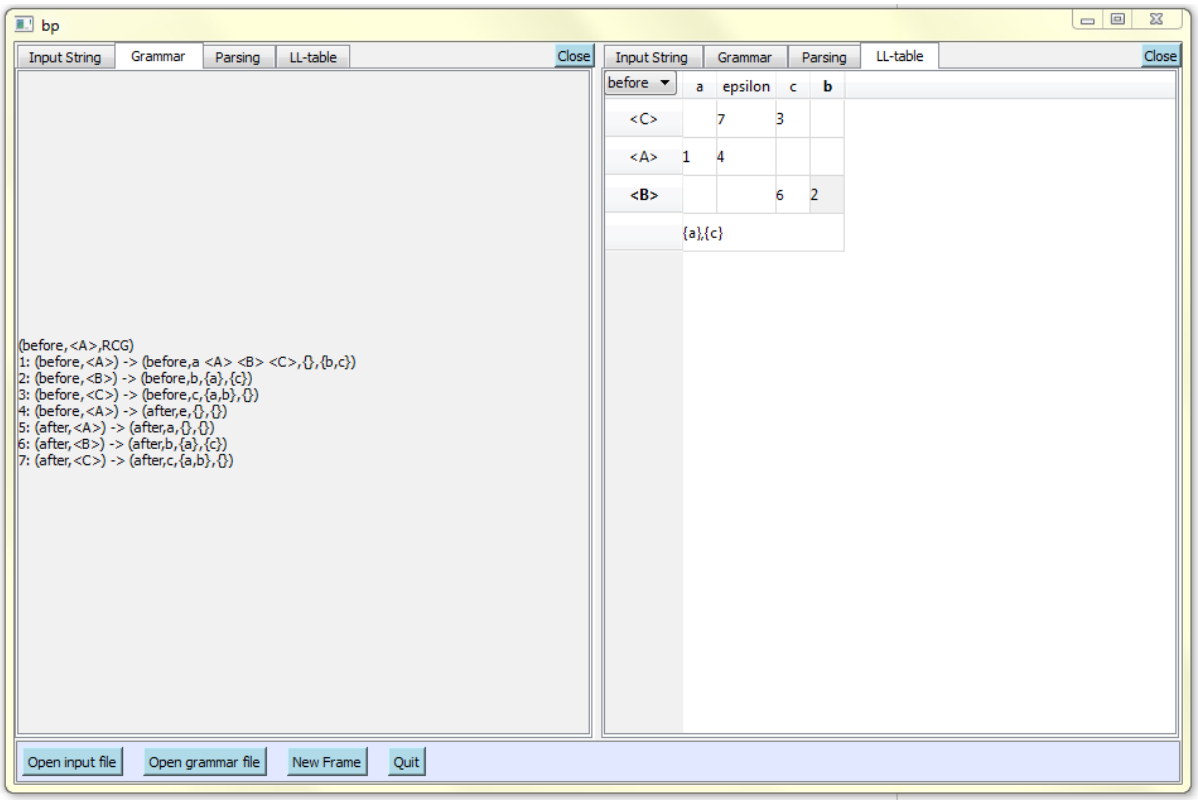
V tomto panelu probíhá krokování běhu syntaktické analýzy, na prvním řádku je vstupní řetězec, na druhém řádku je stav zásobníku v průběhu analýzy. V tomto případě ještě neproběhl ani jeden krok analýzy, na vrcholu zásobníku je počáteční neterminál a vstupní řetězec je zobrazen celý.



Obrázek 5.2: Ukázka panelu `parsing`

Jak je vidět, na pravé straně panelu se nacházejí dvě tlačítka, `From Start` a `Next Step`. První zmíněné vrátí stav krokování do stavu, ve kterém se vyskytoval na začátku. Druhé tlačítko provede jeden krok syntaktické analýzy.

5.3 Panely Gramatika a LL tabulka



Obrázek 5.3: Panely `Grammar` a `LL-table`

Na tomto obrázku vidíme dva panely, v panelu vlevo je zobrazená načtená gramatika. V pravé části obrázku je LL tabulka, která byla vygenerována na základě gramatiky. Jednotlivá políčka tabulky se dají označit pro další informace. V levém horním rohu tabulky se nachází přepínač stavu, pro který platí daná pravidla. Zobrazená tabulka se změní při změně tohoto stavu.

5.4 Vstupní řetězec

Poslední typ obsahu panelu je vstupní řetězec, pokud je vybrán, na obrazovce se objeví v tom tvaru, v jakém byl načten ze souboru.

6 Implementace

Celý program bakalářské práce je implementován v jazyce Python (2.7) a pro zobrazení uživatelského rozhraní je použita knihovna pyQt (4). V rámci implementace byl použit objektově orientovaný přístup.

6.1 Rozdělení programu

Program je modulárně rozdělen na několik částí. Z těchto částí se několik stará o funkci uživatelského rozhraní, jako například položky menu. Uživatelské rozhraní je pro větší přehlednost ještě rozděleno do samostatných souborů podle funkčnosti. Samostatným částem rozhraní odpovídají jednotlivé třídy programu, tyto třídy dědí z tříd existujících v knihovně pyQt. Některé části byly vytvořeny kombinací knihovních tříd. Další část je pro načtení vstupní gramatiky a její převedení do tabulky. Následně je tu modul starající se o syntaktickou analýzu. Poslední dvě části jsou konfigurační soubor sloužící pro snadnější řízení některých částí programu, jako je například počáteční velikost okna, a modul s datovými strukturami, pro uložení například gramatiky nebo načteného vstupního souboru.
Jedním z cílů v rámci implementace byla přehlednost kódu, které bylo dosaženo na základě deskriptivně pojmenovaných metod a proměnných, rozdělení programu na logické celky a komentování složitějších pasáží.

6.2 Konfigurační soubor

Jeden z modulů aplikace je *config.py*, tento modul obsahuje konstanty, které řídí základní prvky uživatelského rozhraní. Před spuštěním programu je v tomto modulu možné upravit vlastnosti jako například velikost písma, výchozí počet panelů nebo cestu označující gramatiku a vstupní soubor načtený při spuštění programu. Tento soubor obsahuje také grafické styly ovlivňující vzhled jednotlivých částí rozhraní.

6.3 Kaskáda při spuštění

Při spuštění aplikace je nejprve vytvořeno hlavní okno programu. Toto okno obsahuje ve výchozím nastavení dva ovládací panely. Při vytvoření panelu je prozkoumán seznam již načtených gramatik a vstupních souborů. Pokud už došlo k načtení gramatiky, jež má být zobrazena v panelu, dojde na povel ze strany kódu ovládajícího panel k předání datového objektu, v němž jsou obsažena všechna důležitá data gramatiky. Pokud však gramatika není uložena v datovém objektu, musí nejprve dojít k jejímu načtení a převedení na LL tabulku. Obdobně probíhá i načtení souboru obsahující vstupní řetězec. Jelikož pro naše účely není třeba žádný pokročilejší lexikální analyzátor, je vstup rozdělen pouze na základě bílých znaků.

6.4 Načtení gramatiky

Zpracování gramatiky ze souboru je iniciováno z řídicí části programu. Tato část je úzce spojena s uživatelským rozhraním. Samotný proces načtení gramatiky začíná otevřením souboru zadaného cestou. Po úspěšném otevření souboru je analyzován první řádek a poté je rozhodnuto, jak se bude ke gramatice přistupovat. Obsahem úvodního řádku souboru jsou povinné informace nutné nejen k rozpoznání, ale také k následnému běhu konverze na LL tabulku.

Příklad: (<A>)
V tomto případě je zadán pouze počáteční neterminál, jedná se tedy o klasickou gramatiku.

Příklad: (base,<var>)
Toto zadání odpovídá stavové gramatice, kromě počátečního neterminálu je určen i počáteční stav.

V obou výše zmíněných případech může být dále přidáno označení `RCG` (zkratka z `Random Context Grammars`), které značí, že bude použito aparátu Gramatik s nahodilým kontextem.

Příklady: (before,<A>,RCG) nebo (<A>,RCG)

Postup při převodu gramatiky na LL tabulku je následující. Ze souboru jsou načtena jednotlivá pravidla, ty jsou rozpoznána pomoci regulárních výrazů a následně, pomocí algoritmu založeném na množinových operacích, převedena. V této části jsou též rozpoznány a zahrnuty do struktury tabulky stavové části pravidel a případně `RCG` pravidla. Tato funkčnost je implementována v modulu *grammarFunctions.py*.

6.5 Syntaktická analýza

V programu je implementována metoda pro jeden krok syntaktické analýzy. Tato metoda je opakovaně volána ze strany uživatelského rozhraní. Funkce se snaží primárně použít striktně nejlevějšího vstupního symbolu a v případě nenalezení pravidla přechází do režimu hledání nejlevějšího možného symbolu. Tento postup je implementován v souboru *parsingModule.py*.

Metoda nejprve zkontroluje, jestli nedošlo k vyprázdnění zásobníku, pokud ano přejde se rovnou k rozhodování, jestli byla analýza úspěšná. Pokud je však zásobník neprázdný, začne metoda postupně provádět krok, v jehož jádru dojde k přijetí terminálního symbolu, rozgenerování neterminálu nebo v případě nenalezení žádného pravidla k ukončení analýzy. V každé z těchto větví také dojde k volání aktualizace dat zobrazovaných v uživatelském rozhraní.
Jestliže je na vrcholu zásobníku terminál, tak se analyzátor pokusí přijmout terminál ze vstupu. Terminál je přijat z místa jeho výskytu. Toto místo nemusí být nutně na začátku řetězce, mohlo totiž dojít k použití nejlevějšího možného hledání pravidla.
V případě neterminálu na vrcholu zásobníku metoda hledá pravidlo na jeho rozgenerování. Pokud nedojde k nalezení pravidla pro striktně nejlevější terminál na vstupu, dojde k průchodu vstupního řetězce a hledání vhodné symbolu tak, aby pro něj a pro neterminál na vrcholu zásobníku odpovídalo nějaké pravidlo. Pokud program při tomto průchodu dojde na konec vstupního řetězce, je za vstup považováno ε. Až v případě, že ani s těmito prostředky není nalezeno vhodné pravidlo, dojde k označení řetězce za nevyhovující, automat jej nepřijímá.

6.6 Grafické rozhraní

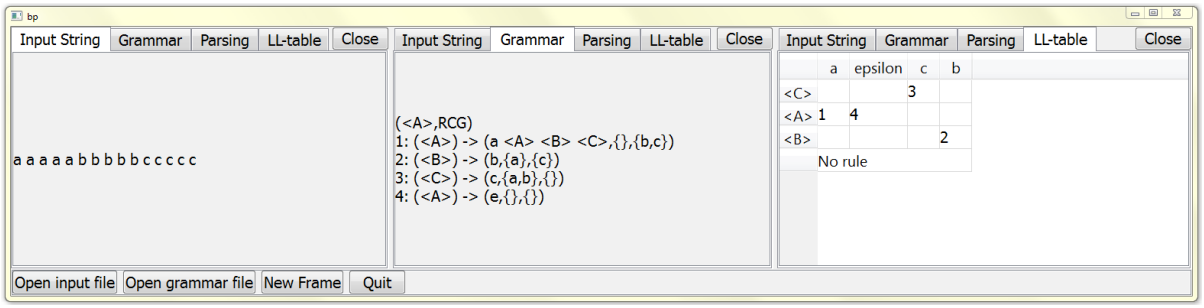
Každá grafická třída v rámci programu dědí z některého základního prvku z knihovny pyQt. Hierarchie tříd tvoří strukturu rozhraní. Celé rozhraní je integrováno do kaskády při spuštění. Vzhled některých komponent společně s nastavením písma se nachází v modulu *config.py*.

7 Výsledky a testování

Nejpodstatnější informace, která plyne z výsledků, je, že samotné stavové gramatiky nejsou při využití LL(1) analýzy dostatečně mocné, aby dokázaly přijímat kontextové jazyky. I proto byla práce rozšířena o přídavné aparáty, které umožnily zvýšit sílu gramatik a dosáhnout původní předpokládané síly.

Program, který je výsledkem práce, je schopný načíst gramatiku a s její pomocí syntakticky analyzovat kontextový jazyk, jenž dané gramatice odpovídá.

Na následujícím obrázku máme příklad spuštění programu pro gramatiku, která obsahuje gramatiku s *RCG* rozšířením. Tato gramatika přijímá pouze jazyk $a^n b^n c^n$.



Obrázek 7.1: Příklad spuštění
Pozn. Velikost písma byla v rámci testování zvětšena, aby byl následný výstup čitelnější.

Součástí testování aplikace bylo i zkoušení kvality uživatelského rozhraní. Hodnotícím faktorem byla hlavně přehlednost dodaných informací, ale také jednoduchost ovládání programu.
Výběr lidí, kteří prováděli testování, byl zkomplikován úzkým zaměřením tématu programu. V rámci testování byl každý, kdo měl alespoň základní znalosti v oblasti formálních jazyků schopen pochopit dodané informace a snadno se v nich zorientovat. Jedním z testovacích podmínek byla také znalost anglického jazyka.
Část testovací skupiny uživatelů navyklá používání typických uživatelských rozhraní byla schopná s programem ihned intuitivně pracovat. Méně zkušeným uživatelům stačilo podat krátké vysvětlení některých principů.

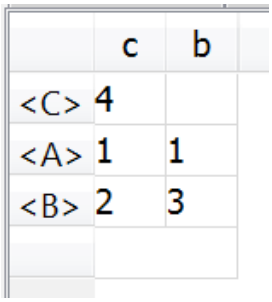
Zde uvádím jednotlivé kroky analýzy na konkrétních příkladech.

Uvažujeme LL(1) gramatiku:

<A>
1: <A> → <B>
2: <B> → <C> <B>
3: <B> → b
4: <C> → c

Této gramatice odpovídá LL tabulka:

	c	b
<A>	1	1
<B>	2	3
<C>	4	



Obrázek 7.2:
Program generuje tuto tabulku v tomto tvaru.

Uvažujeme vstupní řetězec ccb. Samotná analýza pak probíhá takto:

Krok 1.	**c** c b	Začátek syntaktické analýzy.
	<A>	Původní stav.
Krok 2.	**c** c b	Rozgenerování neterminálu <A>.
	<B>	Použití pravidla 1.
Krok 3.	**c** c b	Rozgenerování neterminálu <B>.
	<C> <B>	Použití pravidla 2.
Krok 4.	**c** c b	Rozgenerování neterminálu <C>.
	c <B>	Použití pravidla 4.
Krok 5.	**c** b	Přijetí terminálu 'c'.
	<B>	
Krok 6.	**c** b	Rozgenerování neterminálu <B>.
	<C> <B>	Použití pravidla 2.
Krok 7.	**c** b	Rozgenerování neterminálu <C>.
	c <B>	Použití pravidla 4.
Krok 8.	**b**	Přijetí terminálu 'c'.
	<B>	
Krok 9.	**b**	Rozgenerování neterminálu <B>.
	b	Použití pravidla 2.
Krok 10.		Přijetí terminálu 'b'.

Řetězec byl úspěšně přijat.
Příklad stavové gramatiky.

1,<S>
1: (1,<S>) → (2, a c<A>)
2: (1,<S>) → (3, b c <A>)
3: (2,<A>) → (2, a c <A>)
4: (2,<A>) → (2, ε)
5: (3,<A>) → (3, b c <A>)
6: (3,<A>) → (3, ε)

Této gramatice odpovídají tyto stavové LL tabulky:

1	a	b
<S>	1	2

2	a	ε
<A>	3	4

3	b	ε
<A>	5	6

Automat založen na této gramatice přijímá $(ac)^+ + (bc)^+$.
Uvažujme vstupní řetězec aca.

Krok 1.	**a** c a	Začátek syntaktické analýzy.
	1 | <S>	Původní stav.
Krok 2.	**a** c a	Rozgenerování neterminálu <S>.
	2 | a c <A>	Použití pravidla 1.
Krok 3.	**c** a	Přijetí terminálu 'a'.
	2 | c <A>	
Krok 4.	**a**	Přijetí terminálu 'c'.
	2 | <A>	
Krok 5.	**a**	Rozgenerování neterminálu <A>.
	2 | a c <A>	Použití pravidla 3.
Krok 6.		Přijetí terminálu 'a'.
	c <A>	

Analyzátor již nemůže provést další krok, vstupní řetězec je prázdný, ale na zásobníku zůstalo 'c <A>'. Tento řetězec neodpovídá jazyku, generovaného touto gramatikou.

Další příklad používá stejnou gramatiku jako předchozí.
Tentokrát je vstup bcbc.

Krok 1.	**b** c b c	Začátek syntaktické analýzy.
	1 | <S>	Původní stav.
Krok 2.	**b** c b c	Rozgenerování neterminálu <S>.
	3 | b c <A>	Použití pravidla 2.
Krok 3.	**c** b c	Přijetí terminálu 'b'.
	3 | c <A>	
Krok 4.	**b** c	Přijetí terminálu 'c'.
	3 | <A>	
Krok 5.	**b** c	Rozgenerování neterminálu <A>.
	3 | b c <A>	Použití pravidla 5.
Krok 6.	**c**	Přijetí terminálu 'b'.
	3 | c <A>	
Krok 7.		Přijetí terminálu 'c'.
	3 | **<A>**	
Krok 8.		Rozgenerování neterminálu <A>.
		Použití pravidla 6.

8 Závěr

Při vypracování kvalifikační práce jsem získal několik nových poznatků týkajících se použití stavových gramatik. Například, že jejich kombinace s LL analýzou omezuje jejich sílu. Mohlo by se tedy zdát, že tím nemá toto spojení žádný praktický smysl. Z jiného hlediska však tyto gramatiky přinášejí řadu výhod. První, a dost možná nejdůležitější, výhodou je zvýšení přehlednosti gramatik a syntaktické analýzy. Přítomnost stavů funkčně odděluje části gramatik a LL tabulek a tím zvyšuje jejich čitelnost. Tento význam se dá přirovnat k modulárnímu přístupu v programování. Další výhodou je usnadnění zápisu deterministických pravidel, zvláště za použití gramatik s nahodilým kontextem se může stát, že se pravidla budou v jistých případech chovat nedeterministicky. Tyto případy je možné částečně obcházet za použití stavových pravidel.

Na druhou stranu, stavové gramatiky nijak významně nepřispěly řešení úkolu vytvoření syntaktického analyzátoru použitelného pro kontextové jazyky. Těchto vlastností bylo nakonec dosaženo pomocí jiných rozšíření než stavových gramatik.

Metoda syntaktické analýzy, založena na metodě LL analýzy typu 2, se ukázala jako mocný nástroj z hlediska rozšíření množiny řetězců přijímanými danou gramatikou. Tato metoda ovšem postrádala dostatečné kontrolní mechanismy k dosažení jednoho z cílů, a to vytvoření gramatiky, která by přijímala pouze jazyk $a^n b^n c^n$.

Tento výsledek byl předpokládán na základě generativní síly těchto gramatik. Ta dosti závisí na jejich n-limitovanosti. LL(1) analýza tvrdě omezí sílu těchto gramatik. Použití těchto postupů k tvorbě standardního překladače tedy nebude nejvhodnější volba. Na druhou stranu, s patřičnými rozšířeními, jak ukázal program, mohou tyto postupy získat větší sílu. Tyto vlastnosti by po vhodných optimalizacích mohly být použitelné. Navíc jejich obecnost nabízí snadný postup při vytváření a testování konkrétních příkladů.

Tento problém byl nakonec vyřešen přidáním restriktivních pravidel z gramatik s nahodilým terminálním kontextem. Jelikož gramatiky s nahodilým kontextem kompletně neodpovídaly potřebám analyzátoru, bylo nutné nadefinovat nový pojem. Tímto pojmem jsou gramatiky s náhodným terminálním kontextem.

Program, který je součástí práce, obsahuje všechny původně plánované prvky. Tedy hlavně možnost přehledného krokování postupu syntaktické analýzy, ale také zobrazení dat využívaných programem. V rámci implementace grafického rozhraní jsem se, co se týče panelů, inspiroval moderními internetovými prohlížeči. Okruh lidí, který se účastnil testování rozhraní, jej hodnotil pozitivně.

Jako pokračování práce by bylo možné rozšířit možnosti programu z hlediska uživatelského přístupu. Například zvýšit možnosti krokování pomocí bodů přerušení nebo umožnit upravovat struktury programu za jeho běhu.

Zajímavým rozšiřujícím tématem by také mohlo být větší řízení programu gramatikou. Lexikální analyzátor nebyl součástí této práce, mohl by proto být také řízen gramatikou.

Co se týče oblasti teorie jazyků, práce by mohla pokračovat detailním prostudováním kombinace RCG se stavovými gramatikami, nebo rozšířením popisu kombinace stavových gramatik s LL přístupem o matematické důkazy.

Literatura

[1] AHO, Alfred V. 1997. *Handbook of formal languages: principles, techniques*. 2nd ed. Berlin: Springer Verlag, 625 s. ISBN 35-406-0649-1.

[2] AHO, Alfred V. 2007. *Compilers: principles, techniques*. 2nd ed. Boston: Addison Wesley, 1009 s. ISBN 03-214-8681-1.

[3] MEDUNA, Alexander, ZEMEK, Petr. 2010. *Regulated Grammars and Their Transformation*. První. Brno: Vysoké učení technické v Brně Fakulta informačních technologií. ISBN 978-80-214-4203-0.

[4] MEDUNA, Alexander, ZEMEK, Petr. 2014. *Regulated Grammars and Automata*. New York: Springer. ISBN 978-1-4939-0369-6.

[5] PyQt4 Reference Guide. 2014. *PyQt4 Reference Guide* [online]. [cit. 2015-05-13]. Dostupné z: http://pyqt.sourceforge.net/Docs/PyQt4/index.html

[6] MEDUNA, Alexander, Lukáš VRÁBEL, Petr ZEMEK. 2012. *One-Sided Random Context Grammars* [online]. [cit. 2015-05-14]. Dostupné z: http://www.fit.vutbr.cz/~izemek/grants.php.cs?file=%2Fproj%2F589%2FPresentations%2FPB07-One-Sided-RCG.pdf&id=589

[7] GREIBACH, Sheila, John HOPCROFT. *Journal of computer and system sciences* [online]. New York: Academic Press, 1969, volumes [cit. 2015-06-24]. ISBN 0022-0000. Dostupné z: http://www.sciencedirect.com/science/article/pii/S0022000069800152

[8] MEDUNA, Alexander, Roman LUKÁŠ. *Formální jazyky a překladače* [online]. 2006, 2009 [cit. 2015-06-25].
Dostupné z: https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/IFJ-IT/texts/OporaIFJ.pdf?cid=9356

[9] ČEŠKA, Milan, Tomáš VOJNAR, Aleš SMRČKA. *Teoretická informatika* [online]. 2003 [cit. 2015-06-25]. Dostupné z: http://www.fit.vutbr.cz/study/courses/TIN/public/Texty/oporaTIN.pdf

Příloha 1 - Manuál

Ovládání programu je dostupné prostřednictvím uživatelské rozhraní ovládané především myší. V programu se všechny informace ukazují na otevřených panelech. V každém otevřeném panelu je možné zvolením příslušné záložky vybrat jaká data se v panelu zobrazí, takže je výběr navíc možné kdykoliv změnit. Pokud je pro větší přehlednost vhodné mít otevřeno více panelů, můžeme tak učinit pomocí kliknutí na tlačítko `New Frame`. Pokud chcete nějaký s panelů uzavřít, každý z nich má ve svém pravém horním rohu tlačítko `Close`, které daný panel uzavře.
Tlačítka `Open Input file` a `Open grammar file` slouží k otevření vstupního textu a gramatiky. Stiskem tlačítka `Quit`, dojde k ukončení programu. Posledně zmíněná tři tlačítka jsou spolu s tlačítkem `New Frame` vždy viditelná.

Výběr, který typ informací je zobrazen v panelu, se provede pomocí jednoho z tlačítek v horní části panelu. Může se stát, že když otevřeme velké množství panelů a nebo velmi zúžíme okno programu, tlačítka na výběr informací přestanou být dobře viditelná. Přepínání mezi nimi je však stále možné dosáhnout pomocí malých šipek, které se v takovém případě zjeví v pravé horní části panelu.
V případě typů informací `Grammar` nebo `Input String`, nenabízí panel žádné další interaktivní možnosti, pouze zobrazuje daný obsah. U obsahu `LL-table` je možné označit některou z buněk tabulky, pro případné dodatkové informace. Na závěr, s možností `parsing`, se objeví uvnitř panelu dvě nová tlačítka. Tlačítko `From Start` vrátí analýzu znovu do počátečního stavu, zatímco tlačítko `Next Step` provede další krok syntaktické analýzy.

Příloha 2 – CD

- Veškeré skripty programu
- Složka testovacích vstupů
- Textová část bakalářské práce