Konverze modelů bezkontextových jazyků

Bc. David Navrkal

Diplomová práce
2017

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

*** Nascanované zadání, strana 1 ***

*** Nascanované zadání, strana 2 ***

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky. Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně .....

podpis autora

ABSTRAKT

Tato práce je rozšířením mé bakalářské práce, která se zabývala, jak didakticky prezentovat studentům formálních jazyků jazyky regulární. Proto v této práci budu pokračovat jazyky bezkontextovými. Dočtete se zde o teorii bezkontextových jazyků s množstvím příkladů. Budu se zde zabývat modely bezkontextových jazyků a to zásobníkovým automatem a bezkontextovou gramatikou. U bezkontextové gramatiky bude představena i její nejpoužívanější forma a o je Backusova-Naurova forma. Dále v teorii budou prezentovány algoritmy pro převod bezkontextové gramatiky na zásobníkový automat a zpět. Projektová část se bude zabývat návrhem uživatelského rozhraní pro převodní algoritmy a krátkou rekapitulací programu vyvinutého v rámci bp. práce do kterého tyto převodní algoritmy budu implementovat.

Klíčová slova: Bezkontextové jazyky, zásobníkový automat, bezkontextová gramatika, Backusova-Naurova forma, převody zásobníkového automatu a bezkontextové gramatiky.

ABSTRACT

This work is an extension of my bachalor thesis, which deals with how the didacticly present regular languages to the students of formal languages. Therefore, this work will continue with context-free languages. You can read here about the theory of context-free languages with plenty of examples. I will discuss models of context-free languages such as pushdown automaton and context-free grammar. For context-free grammars will be presented as well its most widely used form the Backus-Naur form. Furthermore, in the theory will be presented algorithms for conversion of context-free grammar to pushdown automata and back. Project part will deal with user interface design for the conversion algorithms and a brief overview on the program developed in bachelor theisis in which I will these algorithms implement.

Keywords: Context-free languages, pushdown automata, context-free grammar, Backus-Naur form, conversions pushdown automata and context-free grammars.

Zde bych chtěl poděkovat za vedení a konzultace mé diplomové préce panu doc. Ing. Romanu Šenkeříkovi, Ph.D.

Také jsem chtěl poděkovat mé přítelkyni za podporu v celém mém studiu a i při tvorbě této práce.

Poslední poděkování patří autorovi této LaTeX šablony diplomové práce, panu Ing. Jozefu Říhovi, která je nyní mnohem přehlednější a navíc se dá mnohem snadněji zkompilovat pomocí pdflatex a také zde můžu používat obrazky ve formátech jpg a png.

OBSAH

ÚVOD

Tato práce pojednává o vývoji aplikace která má didakticky prezentovat studentům teoretické informatiky oblast bezkontextových jazyků. Aplikace navazuje na bakalářskou práci, jejíž praktickou částí bylo implementovat vybrané konverze modelů bezkontextových jazyků. Toto téma jsem si vybral z toho důvodu, že oblast formálních jazyků může být pro studenty hůře pochopitelná a pomocí mé aplikace si tak mohou tuto náročnou teorii procvičit na praktických příkladech.

Zdrojové kódy a text bakalářské práce, na kterou tímto navazuji, jsou dostupné pod open-source licencí na internetovém serveru GitHub. Tato práce společně se zdrojovými kódy zde bude uveřejněna taktéž na `https://github.com/navrkald/regularConvertor`.

Jak již napovídá název serveru Github, práci včetně zdrojových kódů verzuji pomocí systému Git. Díky tomu mám jistotu, že ať se stane cokoliv svoje data mám tímto zálohovaná. Dále pak se mohu v případě potřeby vracet ke starším verzím. Mohu monitorovat svou práci, protože vidím kolik jsem který den udělal. Mohu také nový kód ukládat do větví, což mi umožňuje mít stabilní větev pro koncové uživatele a vývojovou větev, ve které si vyvíjím novou funkcionalitu. Jelikož jsou zdrojové kódy veřejně přístupné, tak v případě zájmu může kdokoliv přispět k v vývoji této výukové aplikace a v případě potřeby může provést takzvaný pull reguest, který mohu schválit a začlenit jej do aplikace.

Pro vývoj tohoto programu jsem si zvolil jazyk C++, protože je mi velmi blízký, jelikož jej používám ve svém povolání programátora ve firmě AVG a také na tento jazyk byl kladen na mé předchozí škole FIT VUT v Brně. Jelikož se pohybuji ve svém životě na pomezí světa Windows a Linuxu, tak pro mě bylo zásadní, aby aplikace fungovala na těchto dvou operačních systémech, tak jsem si zvolil pro vývoj grafického uživatelského prostředí knihovnu Qt verze 5, která je v současné době nejnovější.

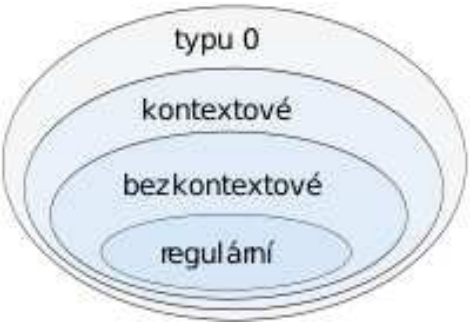
Tuto práci jsem se na rozdíl od své bakalářské, kterou jsem psal v programu Open Office, rozhodl psát v jazyce LaTeX. Toto rozhodnutí má hned několik důvodů. Prvním z nich je to, že jsem se chtěl tento jazyk už dlouho naučit, protože mi je jako programátorovi mnohem bližší. Nejen technické texty vypadají mnohem lépe v tomto jazyce, protože se za mě stará za základní typografická pravidla a také mám při psaní v LaTeXu větší kontrolu nad textem. Dalším důvodem, je že pomocí jazyka LaTeX můžu mít práci napsánu v textovém formátu, na rozdíl od programů Open Office nebo Microsoft Word, které používají binární formát, který si nemohu otevřít v poznámkovém bloku, nebo nad ním dělat jednoduše porovnání dvou verzí. Tato zkutečnost znamená i mnohem efektivnější verzování jednotlivých commitů v Gitu, ve kterém si pak můžu prohlížet a porovnávat jednotlivé verze a případně se i vrátit k některé z předchozích verzí [1].

TODO: Psat uvod smerem, co bylo cílem aplikace a co by měla umět.

I. TEORETICKÁ ČÁST

1 Chomského hierarchie

Abychom si mohli dát bezkontextové jazyky do souvislosti s ostatními formálními jazyky, je nutné si vysvětlit Chomského hierarchii. Tato hierarchie byla vytvořena panem Noamem Chomskym v roce 1956. Dává do souvislosti jazyky regulární, bezkontextové, kontextové a jazyky typu 0. Jejich vztah, můžete vidět na obrázku (Obr. 1.1), jako vztah množin.



Obr. 1.1 Chomskeho hierarchie formálních jazyků

Jak je vidět z obrázku regulární jazyky zaujímají nejmenší množinu, nebo chcete li nejmenší počet jazyků. Nenechte se však zmýlit pojmem počet jazyků, protože počet jazyků a množství řetězců generovaných daným jazykem jsou dvě rozdílné věci. Například jednoduchým regulárním výrazem $(a+b)^*$ lze popsat všechny slova nad abecedou $\Sigma = a, b$. Avšak jazyk, všech slov, kde první půlka slova se skládá ze stejného počtu písmen 'a' jako písmen 'b' v druhé půlce slova regulárním jazykem nepopíšeme. Dalo by se tak říci, že čím výše se jazyk nachází v Chomského hierarchii, tím přesněji dokážeme jazyk popsat, jinými slovy, tím více jazyků jsme schopni generovat.

Nyní si pro lepší představu u každého jazyku uvedeme modely, které daný jazyk popisují s jedním příkladem konkrétního jazyka.

Regulární jazyky můžeme popsat pomocí regulárních gramatik, regulárním výrazem nebo konečným automatem. Příkladem je například jazyk který obsahuje libovolnou kombinaci písmen 'a' a 'b' končící písmenem 'b'. Tento jazyk by se dal pospat regulárním výrazem $(a+b)^*b$.

Jazyky bezkontextové lze popsat modely jakými jsou bezkontextové gramatiky a konečný zásobníkový automat. Všechny jazyky regulární jsou zároveň bezkontextové, avšak obrácené tvrzení neplatí. Příkladem je jazyk $L = \{a^n b^n : n >= 1\}$

Jazyky kontextové můžeme popsat lineárně ohraničeným Turingovým strojem. Příkladem nechť je jazyk $L = \{a^n b^n c^n : n >= 1\}$

TODO: Doplnit jazyk typu 0.

Třída jazyků typu nula, v sobě obsahuje všechny doposud zmíněné jazyky plus ještě jazyky navíc. Popisujeme je úplným Turingovým strojem. [2]

2 Bezkontextový jazyk

V předchozí kapitole jsme si uvedli vzájemný vztah formálních jazyků. Nyní se pojďme podívat podrobněji na bezkontextové jazyky. Tyto jazyky popisujeme pomocí dvou hlavních modelů, kterými jsou bezkontextová gramatika (CFG) a konečný zásobníkový automat (PDA).

Nenechme se však zmýlit představou, že pomocí obou můžeme generovat bezkontextové jazyky. Protože pomocí bezkontextové gramatiky můžeme jazyk generovat, avšak pomocí konečného zásobníkového automatu můžeme rozpoznat, zda konkretní jazyk je popsán daným automatem.

Pro nás je každopádně důležitá zkutečnost, že oba popisy jsou vzájemně ekvivalentní, tj. že lze převést PDA na CFG a taktéž lze převést CFG na PDA. Formální popis bezkontextové gramatiky a zásobníkového automatu bude uveden dále spolu s jejich vzájemnými konverzemi.

Pro lepší představu jaké jsou to vlastně ty bezkontextové jazyky si nyní uvedeme pár příkladů. Krásným příkladem z praxe by mohl být jazyk závorek, tak aby odpovídal počet levých závorek počtu závorek pravých. Složitějším příkladem by mohl být jazyk matematicky správných algebraických výrazů, který se skládá z operátorů plus, mínus, krát, děleno, symbolů 'x', 'y', 'z' a závorek. Dalším příkladem z praxe je, že velká podmnožina programovacích jazyků se dá popsat pomocí bezkontextové gramatiky.

Jak je vidět z těchto příkladů, pomocí bezkontextového jazyka se dají popsat složitější věci a jejich uplatnění můžeme nalézt u popisu programovacích jazyků. Existuje notace bezkontextové gramatiky, zvaná Backusova-Naurova forma, která se právě často používá pro popis syntaxe programovacích jazyků. Znalost bezkontextových jazyků lze využít při konstrukci překladače, nebo interpretu programovacího jazyka a také při počítačovém zpracování algebraických výrazů. [3]

3 Zásobníkový automat

Nyní si definujeme definici zásobníkového automatu. [4]

Konečný zásobníkový automat M je uspořádaná sedmice
$M = (Q, \Sigma, \Gamma, R, q_0, S, F)$ kde:

- Q je konečná množina vnitřních stavů,
- Σ je konečná vstupní abeceda,
- Γ je konečná abeceda zásobníku,
- R je přechodová relace, popisující pravidla činnosti automatu, je definovaná jako konečná množina kartézského součinu $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times Q \times \Gamma^*$,
- q_0 je počáteční stav z množiny Q,
- S je počáteční symbol na vrcholu zásobníku z množiny Γ,
- F je množina koncových stavů, $F \subseteq Q$.

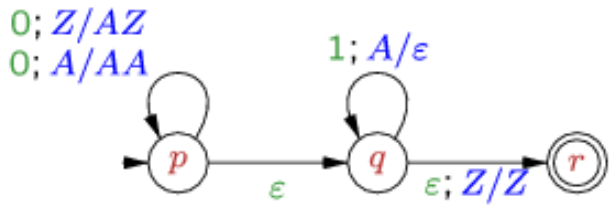
Definice 1 *Definice zásobníkového automatu.*

Pro lepší představu si uvedeme příklad zásobníkového automatu M, který popisuje jazyk $L = \{0^n 1^n$, kde $n \geq 0\}$.

Nechť M je definován následovně:

- $Q = \{p, q, r\}$,
- $\Sigma = \{0, 1\}$,
- $\Gamma = \{A, Z\}$,
- $R = (p, 0, Z, p, AZ), (p, 0, A, p, AA), (p, \epsilon, Z, q, Z), (p, \epsilon, A, q, A), (q, 1, A, q, \epsilon), (q, \epsilon, Z, r, Z),$
- $q_0 = p$,
- $S = Z$,
- $F = \{r\}$.

Graficky znázorněný stejný zásobníkový automat by vypadal následovně:



Obr. 3.1 Příklad zásobníkového automatu

3.1 Šipková notace přechodových pravidel zásobníkového automatu

Pro větší přehlednost a čitelnost přechodových pravidel zásobníkového automatu zde uvedu definici přechodových pravidel, kterou budu používat dále v této práci.

Přechodová pravidlo $r \in R$ konečného zásobníkového automatu
$M = (Q, \Sigma, \Gamma, R, q_0, S, F)$

Definujeme následovně:
$r = Apa \rightarrow wq$, kde p a $q \in Q$, $A \in \Gamma$, $a \in \Sigma$ a w je řetězec symbolů náležících do Γ.

Definice 2 *Definice přechodových pravidel zásobníkového automatu v šipkové notaci.*

4 Bezkontextová gramatika

Definice bezkontextové gramatiky je následující.

Bezkontextová gramatika G je uspořádaná čtvrtice $G = (V, \Sigma, R, S)$ kde:

- V je konečná množina non-terminálních symbolů,
- Σ je konečná množina terminálních symbolů,
- R je konečná množina relací z V do $(V \cup \Sigma)^*$, taktéž se nazývá množinou přepisovacích pravidel,
- $S \in V$ se nazývá počáteční non-terminální symbol gramatiky.,

Definice 3 *Definice bezkontextové gramatiky.*

Jednotlivé přepisovací pravidla často píšeme pomocí „šipkové notace“, kde na levé straně neterminál za ním následuje šipka a na pravé straně je řetězec non-terminálních symbolů. Přepisovací pravidla se nazývají protože, řetězce daného jazyka, které jsou popsány gramatikou vznikají tak, že na počátku máme řetězec skládající se z počátečního non-terminálního symbolu gramatiky S a na ten aplikuji jedno z přepisovacích pravidel. Pokračuji tím, že postupně nahrazuji non-terminální symboly v řetězci do té doby, dokud mě v něm nezbudou jen terminální symboly a to je pak jedno konkrétní slovo daného bezkontextového jazyka.

Například gramatika $G = (\{S, B\}, \{a, b\}, R = \{\{S \to aBb\}, \{B \to aBb\}, \{B \to \varepsilon\}\}, S\})$ popisuje jazyk $L = \{a^n b^n : n >= 1\}$.

Příklad konkrétního slova, které generuje tato gramatika by mohl vypadat následovně:

1. Na začátku máme slovo skládající se z počátečního symbolu S.
2. Po aplikaci přepisovacího pravidla $S \to aBb$ dostaneme slovo aBb.
3. Toto pravidlo aplikujeme ještě jednou a dostáváme slovo $aaBbb$.
4. Nakonec aplikujeme pravidlo $B \to \varepsilon$ a dostáváme slovo $aa\varepsilon bb$.
5. Jelikož ε značí prázdný řetězec, tak výsledné slovo jazyka L je $aabb$. Slovo se nám skládá jen z terminálních symbolů, takže jsme s generováním slova u konce.

[5]

4.1 Backusova-Naurova forma

Pro zápis bezkontextové gramatiky se používají převážně dvě notace a to Backusova-Naurova forma (BNF) a jako druhá v pořadí je Van Wijngaardenova gramatika. My si zde představíme pouze první z nich, protože s BNF se můžete setkat častěji. Jazyky, které jsou popsány BNF a rozšířenou BNF jsou Algol, Paskal, C/C++, Ada 95, PL/I a další.

TODO: Dodat citaci.

Syntaktická přepisovací pravidla jsou definována tak, že levou stranu vždy tvoří non-terminál, za kterým následuje oddělovač „::=“ (dvě dvojtečky následované rovnítkem) za kterým následuje pravá strana pravidla, která se skládá z řetězců non-terminálů a terminálů. Pravá strana se může skládat z více řetězců oddělených symbolem „|“ značící logický operátor OR. Tento operátor byl zaveden, kvůli optimalizaci zápisu pravidel, tak aby místo zápisu, že jeden non-terminální symbol se může rozepsat na N řetězců, což by muselo být rozepsáno na N pravidel na N řádcích. Místo toho se těchto N řetězců napíše na pravou stranu oddělených symbolem „|“. Terminální symboly jsou uvedeny v uvozovkách. [6]

Gramatika pro popis číselné konstanty by mohla vypadat takto:
<**konstanta**> ::= <**číslice**> | <**konstatnta**> <**číslice**>
<**číslice**> ::= „0“ | „1“ | „2“ | „3“ | „4“ | „5“ | „6“ | „7“ | „8“ | „9“

Gramatika který by popisovala základní aritmetické operace by měla navíc tato pravidla:
<**výraz**> ::= <**výraz**> „+“ <**výraz**>
<**výraz**> ::= <**výraz**> „-“ <**výraz**>
<**výraz**> ::= <**výraz**> „×“ <**výraz**>
<**výraz**> ::= <**výraz**> „/“ <**výraz**>
<**výraz**> ::= „(“ <**výraz**> „)“
<**výraz**> ::= <**konstanta**>

5 Algoritmus převodu CFG na PDA

Nyní si pojďme ukázat algoritmus převodu bezkontextové gramatiky na zásobníkový automat. [7]

Vstup: CFG $G = (V, \Sigma, R, S)$
Výstup: Nedeterministický PDA $M = (Q, \Sigma, \Gamma, R, q_0, S, F)$, který přijímá vstupní řetězec vyprázdněním zásobníku.

1. $Q = q_0$
2. $\Sigma = \Sigma$
3. $\Gamma = V \cup \Sigma$
4. Počáteční stav je q_0
5. Konstrukce přechodové množiny R:
   (a) Pro každé $a \in \Sigma$: přidej $(q_0, a, a, q_0, \varepsilon)$ do R
   (b) Pro každé $A \to x$, kde $A \in V, x \in (V \cup \Sigma)^*$, přidej do $R(q_0, \varepsilon, A, q_0, y)$, kde $y = reversal(x)$
6. $S = S$
7. $G = \varnothing$

Algoritmus 1 *Převod CFG na PDA.*

Po aplikaci tohoto algoritmu nám vznikne nedeterministický PDA s jedním vstupním stavem a žádným ukončovacím stavem. Tento automat přijímá vstupní řetězec vyprázdněním zásobníku. Má stejnou vstupní abecedu, jako byla abeceda terminálů u původní CFG. Zásobníková abeceda automatu vznikla sloučením množin terminálů a non-terminálů vstupní gramatiky. Počátečním symbolem na zásobníku je počáteční non-terminální symbol gramatiky.

První část množiny přechodových pravidel vznikla tak, že pro všechny terminální symboly se přidala přechodová pravidla taková, že počáteční i koncový stav pravidla je vždy počáteční stav automatu. Zároveň aktuální symbol na pásce a stejně tak na vrcholu zásobníku musí být stejný terminální symbol. Znak ε říká, že symbol z vrcholu zásobníku odstraním.

Zbytek přechodových má taktéž počáteční a koncový stav stejný. ε zde říká, že nečtu žádný ze symbolů na pásce, ale přečtu jeden non-terminál z vrcholu zásobníku a nahradím jej řetězcem znaků z pravé části pravidla napsaných pozpátku.

6 Základy Qt frameworku

Qt framework je multiplatformní, to znamená, že programátor napíše jeden kód, který pak zkompiluje na více cílových systémů. Konkretně *Regular Convertor* jsem úspěšně zkoušel na platformách Windows a Linux. Tento framework je určen především pro jazyk C++. Obsahuje také IDE pojmenované QtCreator, ve kterém je i integrován nástroj Qt Designer pro tvorbu uživatelského rozhraní vizuální formou a ne jenom pomocí kódu.

6.1 QWidget

QWidget je základní stavební element a zároveň předek v hierarchii tříd všech grafických elementů uživatelského rozhraní. Například widget je předek všech tlačítek, oken, kontrolních prvků, dialogů, prostě všeho. Samotný widget sám o sobě nemá význam, ale definuje a zaštiťuje společnou množinu vlastností pro všechny od něj odvozené třídy.

6.2 Popis uživatelského rozhraní pomocí ui souborů

TODO: Popsat jak funguje systém UI souborů. Jak se z nich generuje exekutivní kód. Obrázek jak vypadá Qt designer.

6.3 Promoting

Spolu s tím, že QWidget je původní předek všech grafických elementů a proto je předek i všech mých uživatelských elementů jsem mohl v QtDesigneru použít pro mě novou techniku *promoting*. Volně do češtiny by se to dalo přeložit jako povyšování widgetů. Pro všechny konverze jsem si následně vytvořil widgety, které se skládají z jiných také mnou vytvořených widgetů. Proto jsem si vytvořil rozložení widgetů, které jsem následně povýšil na mnou skutečně předem vytvořené widgety. Má to tu nespornou výhodu, že jsem se zbavil všude v kódu, programového vytváření obrazovky pro jednotlivé konverze, které jsou nově definovány jako samostatné třídy, které zapouzdřují z grafického klediska všechny konverze. Tímto, že se popis UI přesunul do samostatných souborů, se značně ulevilo třídě pro popis hlavního okna aplikace. Tato objektově orientované technika rozdělení aplikace do více menších částí a rozdělení pravomocí zvaná zapouzdření.

6.4 Signály a sloty

Signály a sloty se podobá jiné programovací technice zvaná *callbacks*, kde se předává ukazatel na funkci nebo metodu.

Slot je metoda třídy, která se vykoná pokud někdo emitoval signál se kterým je spojená pomocí funkce *connect()*. Stejně jako můžeme propojit emitování signálu s vykonáváním určité metody zvané slot, můžeme později zavolat i opačnou metodu *disconnect()* , která toto spojení zruší.

K jednomu signálu může být propojeno i více slotů. Tak jsem například implementoval, že konečný automat má více grafických reprezentací, tak aby se na základě emitování signálu o změně konečného automatu změnily obě jeho vizuální reprezentace.

Slot musí mít definované svoje tělo, které se vykonává při emitování signálu, pokud jsou propojeni. Naopak tělo signálu nesmí být nikdy definováno.

Signály a sloty si mohou vyměňovat informace pomocí parametrů. V nižších verzích Qt fungovalo propojení na základě maker *SIGNAL* a *SLOT*. V aktuální verzi Qt se může použít i systém ukazatelů na funkce, které jsem použil na propojení konverzních widgetů s hlavním oknem aplikace.

II. PRAKTICKÁ ČÁST

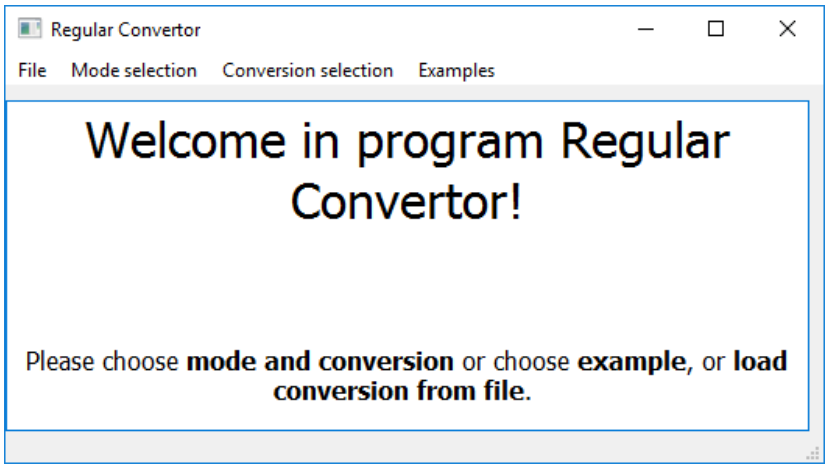
V této části své diplomové práce vám představím aplikaci Regular Convertor z uživatelského hlediska, z pohledu chování, ovládání a funkcionalit. Následně v Implementační části se podíváme blíže na implementaci.

Na úvod jsem chtěl zmínit, že samotnou aplikaci jsem lokalizoval do anglického jazyka, protože jsem se nechtěl omezovat pouze na česky mluvící studenty. Texty z aplikace zde budu citoval anglicky, ale popisovat je budu v češtině.

Aplikace by měla sloužit k účelům prezentace formálních jazyků na přednáškách a zároveň i k samostatnému procvičení studenty. Proto obsahuje část, kde se prezentuje jak vlastní algoritmus funguje, tak i části kde si student nasbíranou teorii prakticky vyzkouší. Důležité je, aby uživatel mohl s danými algoritmy experimentovat, proto program umožňuje vložení libovolného vstupu, s následnou kontrolou jeho validity. Tento vstup si může uživatel vložit ručně a nebo jej načíst ze souboru. Proto lze všechny konverze a jejich módy, včetně vstupních i výstupních dat ukládat a načítat ze souborů. Aplikace obsahuje i řadu zabudovaných příkladů, které jsou členěny od jednodušších ke složitějším.

7 Úvodní obrazovka

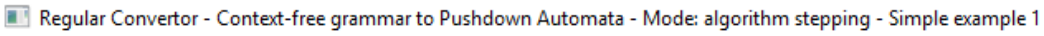
Na obrázku (Obr. 7.1) je pro uživatele připraveno uvítání a zároveň instrukce jak dále postupovat. Uživatel má možnost načíst si již předem připravenou konverzi ze souboru, nebo si otevřít příklad konkrétní konverze a na závěr má možnost si zvolit vlastní konverzi, které předá vstupní data.



Obr. 7.1 Úvodní obrazovka programu Regular Convertor

V záhlaví hlavního okna se zobrazují informace pro snadnější orientaci uživatele, který zde vidí, že spuštěný program se jmenuje "Regular Convertor", dále pak jakou konverzi má vybranou, následuje pak jaký konverzní mód má zvolen a jako poslední je uveden název příkladu, který si uživatel vybral. Pokud si nezvolil žádný z příkladů, tak se jeho název nezobrazuje.

Příklad takového záhlaví hlavního okna, kde je zvolena konverze bezkontextové gramatiky na zásobníkový automat, vybrán je mód krokování algoritmu a první příklad, můžete vidět na obrázku (Obr. 7.2).



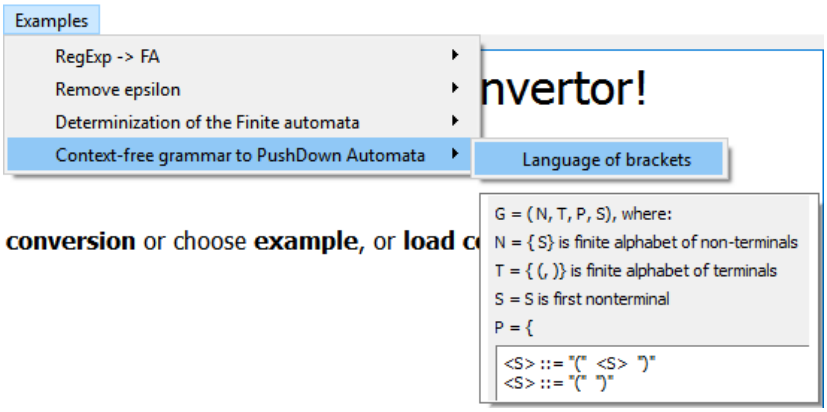
Obr. 7.2 Výřez záhlaví hlavního okna

8 Možnosti začátku práce s programem

V předchozí části bylo popsáno hlavní okno spolu s instrukcemi pro uživatele, jak může začít pracovat s programem. Nyní si je pojďme popsat podrobněji.

8.1 Výběr z příkladů

Příklady pro konverzi bezkontextové gramatiky na zásobníkový automat jsou dostupné v menu pod `Examples -> Context-free grammar to PushDown Automata -> <Název příkladu>`. Po najetí myši na libovolný příklad se zobrazí dole pod položkou menu obrázek vstupních dat pro snadnější orientaci uživatele. Toto ukázání obrázku nad položkou v menu není standardní vlastností Qt frameworku a musel jsem ji ručně naprogramovat. Více se dozvíte dále v části nazvanou implementace. Celou situaci můžete vidět na obrázku (Obr. 8.1).



Obr. 8.1 Snímek obrazovky s výběrem příkladu bezkontextové gramatiky na zásobníkový automat

TODO: Doimplementovat více příkladů a aktualizovat obrázek.

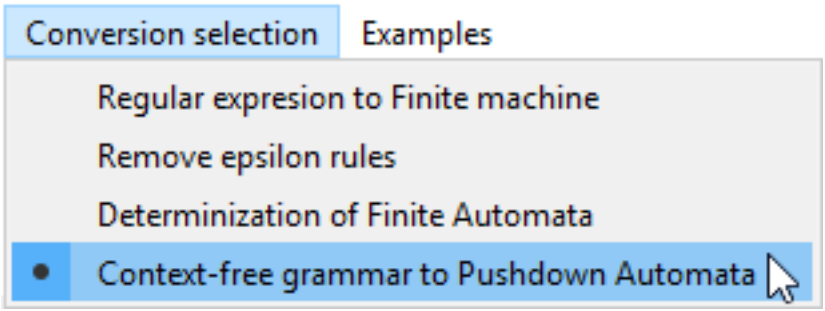
8.2 Výběr z konverzí

Třetí a poslední možností, jak začít používat program je volba konverze. Uživatel má na výběr celkem z 4 konverzí, z čehož poslední z nich a to převod bezkontextové gramatiky na zásobníkový automat byla implementována v rámci této práce. Pokud uživatel využije této volby musí počítat s tím, že vstupní data budou prázdná.

Všechny implementované konverze zobrazuje obrázek (Obr. 8.2).

8.3 Ukládání a načítání konverzí

Tyto volby pro načítání z, respektive ukládání do souboru se nachází v menu pod `"File -> Save"`, respektive pod `"File -> Load"`. Taktéž jsou v aplikaci implemento-



Obr. 8.2 Výseč snímku z obrazovky pro výběr konverzí

vané zkratky `ctrl + o` respektive `ctrl + s` pro otevření, respektive uložení aktuální konverze do souboru. Tuto funkcionalitu jsem přidal zejména z důvodu toho, že zabudované příklady v aplikaci nemohou obsáhnout vše a proto má vyučující možnost připravit další zajímavé příklady pro studenty.

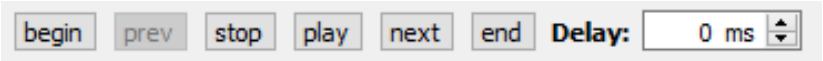
Screenshot není potřeba přikládat, avšak za zmínku stojí fakt, že u módů "Samostatná práce" a "Automatická kontrola" se ukládají data, jak vstupní, tak výstupní, avšak u módu "Krokování algoritmu" se ukládají jenom vstupní data, protože výstup zde netvoří uživatel, nýbrž samotná aplikace.

9 Konverzní módy

Poslední položkou v menu je výběr módu. Jak již bylo řečeno na začátku této kapitoly, aplikace by měla umožňovat, jak samotné studium konverzních algoritmů, tak jejich samostatné praktické procvičení, proto jsem implementoval následující módy.

9.1 Mód krokování algoritmu

První mód určený pro prezentace v hodinách na fakultě a pro samostudium implementovaných konverzí je mód nazvaný *Krokování algoritmu*. V tomto módu si uživatel krokuje jednotlivé kroky algoritmu. Ve widgetu pro danou konverzi jsou v jeho horní části následující kontrolní prvky, které můžete vidět na obrázku (Obr. 9.1). Ještě než se přesuneme k popisu jednotlivých tlačítek stojí za zmínku, že pokud je algoritmus na úplném začátku, tak je zakázáno tlačítko zpět, stejná situace nastává na konci algoritmu pro tlačítko vpřed. První případ je ostatně vidět na obrázku (Obr. 9.1).



Obr. 9.1 Výřez konrolních prvků z krokovacího módu

Jak je vidět, k dispozici jsou možnosti `next`, respektive `prev`, které nás posunují o jeden krok v algoritmu vpřed, respektive v zad. Dalšími jsou `begin`, respektive `end`, které za nás okamžitě přenesou v algoritmu na začátek, respektive na konec. Poslední dvě možnosti jsou `play` a `stop`. Předposlední tlačítko je nazváno `play` začne přehrávat algoritmus po jednotlivých krocích s časovou prodlevou specifikovanou v pravé části pomocí kontrolky pojmenované `Delay`, ve které se zadává čas v milisekundách. Výchozí volba je 0 ms, která říká programu, že má provádět jednotlivé kroky algoritmu jak nejrychleji dovede. Poslední tlačítko je nazváno `stop`, které nám přestane přehrávat algoritmus, pokud jsme předtím klikli na `play`. Více o algoritmickém widgetu a souvislosti breakpointů a tlačítka `play` si povíme dále.

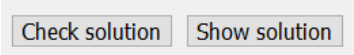
9.2 Mód samostatné práce

Poté co si uživatel prostudoval v praxi, jak funguje algoritmus převodu, si může zkusit konverzi provést sám, tak aby se přesvědčil, že algoritmus plně ovládá.

Poté co ve výstupním widgetu uživatel vytvořil svoje řešení, může si manuálně zkusit, jestli správně převedl daný model formálního jazyka pomocí tlačítka `Check solution`.

Pokud uživatel zjistil, že nevytvořil správné řešení a už neví jak dále, může si nechat správné řešení zobrazit pomocí tlačítka `Show solution`. Po jeho stisku se ve výstupním widgetu zobrazí vzorové řešení. Zároveň s tím se změní text tohoto tlačítka na `Back`, po

jehož stisku se do výstupního widgetu vloží zpět uživatelovo řešení, přesně tak jak jej zanechal před stiskem `Check solution`. Zobrzení techto dvou respektive tří (s právě skrytým tlačítkem zpět) kontrolních prvků můžete vidět na obrázku (Obr. 9.2).



Obr. 9.2 Výřez konrolních prvků z módu samostatné práce

9.3 Mód automatické kontroly

Pokud už si uživatel myslí, že "už už"přece řešení musí být správné, nebo ho jen nebaví manuálně klikat, jestli má už správné řešení, může využít poslední mód pojmenovaný v programu `Instant checking`, taktéž volně přeložený do češtiny *Mód automatické kontroly*. V tomto módu program sám ve velmi krátké periodě kontroluje, jestli je uživatelův vstup ekvivalentní se správným řešením. Více o tom, co je to ekvivalentní řešení, si povíme dále v části popisující převod bezkontextové gramatiky na zásobníkový automat. Stejně jako v předchozím módu se správnost řešení objevuje v levém horním rohu výstupního widgetu.

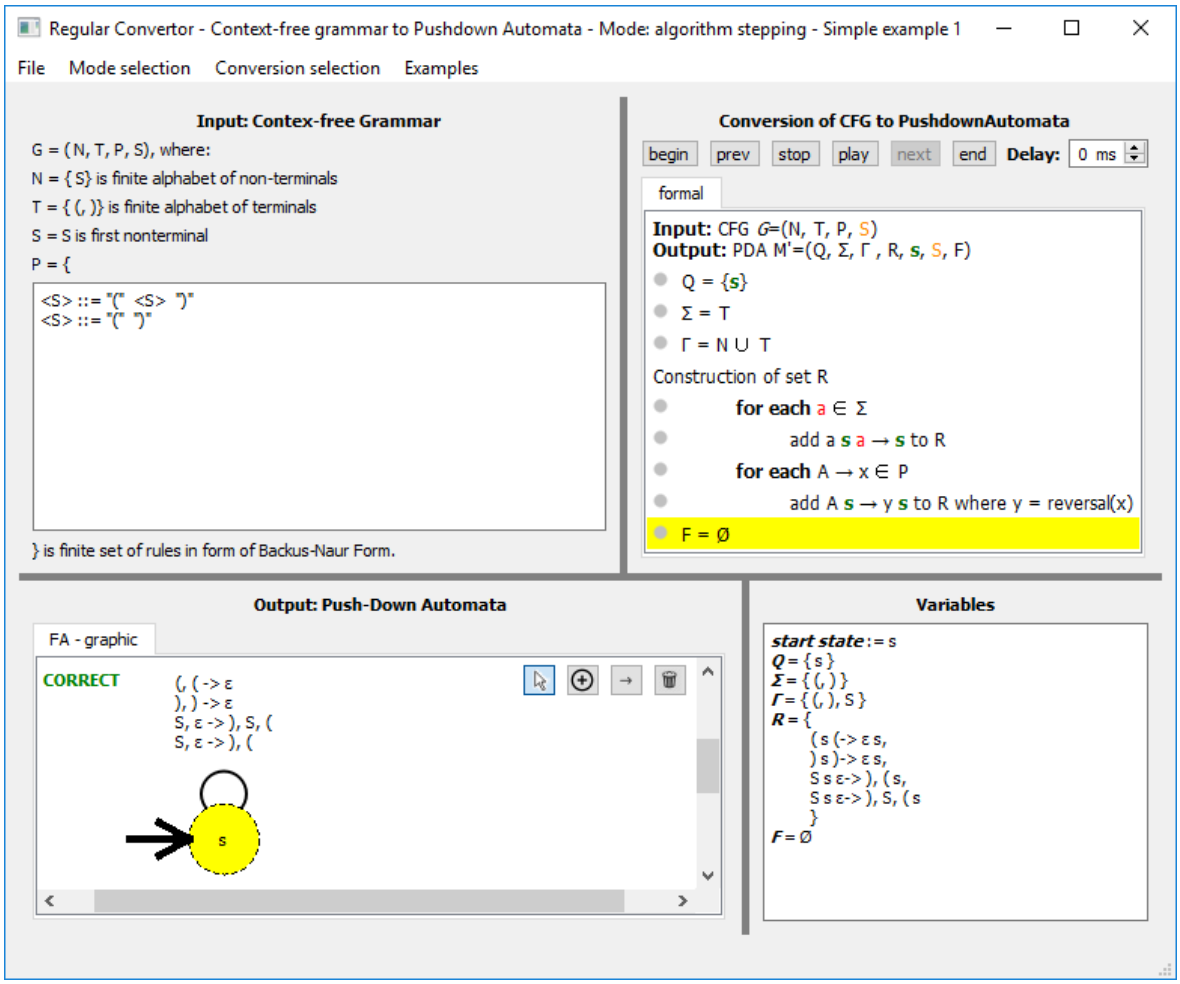
10 Popis konverze bezkontextové gramatiky na zásobníkový automat

Stejně jako u předchozích konverzí, jsem se i v této konverzi držel zavedeného schématu z bakalářské práce, že vlevo nahoře je widget, ve kterém je popsán vstupní model formálního jazyka, v pravé horní části je algoritmický widget, v levé dolní části je widget, ve kterém je umístěn výstupní model formálního jazyka a konečně v pravé dolní části je widget, ve kterém jsou zobrazené aktuální hodnoty proměnných použitých v algoritmickém widgetu. Celý pohled je ukázán na obrázku (Obr. 10.1).

Všechny widgety jsou od sebe navzájem odděleny rozdělovači, které umožňují měnit jejich velikost a to až do té míry, že je možné je úplně schovat. To se může hodit v situaci, když si chce student procvičit danou konverzi do té míry, že nechce být rušen samotným algoritmem. Další ze situací je, když studuje, nebo samostatně provádí konverzi a nezajímají jej aktuální hodnoty proměnných, nebo chce jen více místa pro výstupní widget, pokud převádí rozsáhlý model, tak jistě od-cenní možnost, že si může nechat schovat widget s proměnnými.

V této konverzi vstupní widget popisuje bezkontextovou gramatiku a ve výstupním se nachází zásobníkový automat.

Při kontrole, jestli je uživatelský zásobníkový automat ekvivalentní se správným řešením, nezáleží na tom jak si uživatel pojmenoval jednotlivé stavy a také nezáleží na tom v jakém pořadí uživatel vloží přechodová pravidla.



Obr. 10.1 Pohled na hlavní okno, ke je vybrán první příklad převodu bezkontextové gramatiky na zásobníkový automat na konci krokovacího módu

11 Vstupní widget bezkontextové gramatiky

Do tohoto widgetu se vkládá popis bezkontextové gramatiky pomocí Backus-Naurovi formy popsané v teoretické části této práce. Tento widget je inteligentní a pozná, jestli je daná gramatika ve správném syntaktickém formátu. Pokud není, tak pozadí editačního kontrolního prvku, kde se daná gramatika vyplňuje zežloutne, čímž uživateli signalizuje, že udělal chybu v syntaxi. V opačném případě se nastaví pozadí na bílou barvu, gramatika automaticky rozparsuje a vyplní se tak počáteční non-terminál, abeceda terminálů a non-terminálů. Uživatel záměrně nemá možnost tyto 3 informace editovat, tak aby se předešlo zbytečným chybám. Za zmínku stojí, že pokud mají jednotlivá gramatická pravidla stejnou levou stranu, lze je zapsat na jeden řádek tak, že pravé strany jsou od sebe odděleny svislítkem. Pokud uživatel vybere čistou konverzi s prázdnou bezkontextovou gramatikou, widget je předvyplněn textem, který uživateli názorně ukáže příklad správné formy, jak zapsat bezkontextovou gramatiku. Je zde uveden proto, aby bylo pro nové uživatele co nejsnadnější používat tento program.

Jeho text je uveden na obrázku (Obr. 11.1).

Obr. 11.1 Předvyplněný text bezkontextové gramatiky

12 Algoritmický widget

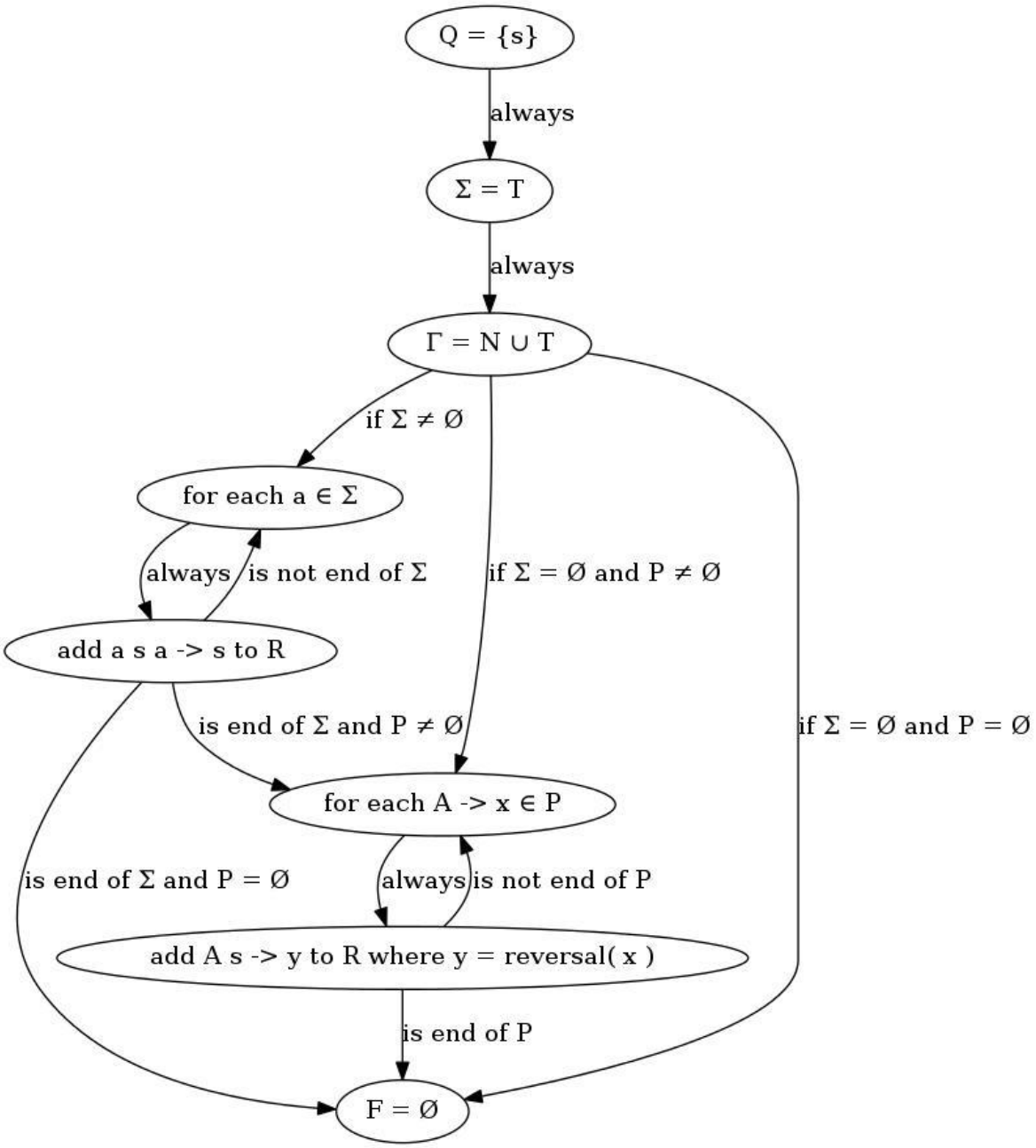
Každý algoritmický widget na samotném vrcholku obsahuje tlačítka a kontrolní prvky, které se mění v návaznosti na vybraný mód. Jelikož jsem je popisoval výše, proto je zde nebudu blíže rozebírat.

Pod nimi se nachází samotný textový popis algoritmu, kde v jeho vrchní části je zapsáno, co je jeho vstupem a výstupem. Pod tímto popisem jsou už samotné kroky algoritmu. Při samotném krokování se aktuální krok vždy zvýrazní žlutým pozadím.

Výpočetní model každého algoritmu je postaven tak, že kliknutím na tlačítko `next` se následující krok provede, jeho pozadí zežloutne, aktualizuje výstupní widget a nastaví aktuální proměnné do widgetu vpravo dole.

Vlevo od exekutivních kroků je šedé kolečko, které se změní na červené, pokud uživatel kliknul na příslušný řádek. Toto kolečko symbolizuje breakpoint a jeho barva značí, jestli je neaktivní, či aktivní. Breakpointy slouží v režimu krokování algoritmu, kde lze kliknout na tlačítko `play` a tím se spustí automatické krokování algoritmu s danou prodlevou mezi kroky do doby, než algoritmus narazí na aktivní breakpoit, nebo dokud nebylo stisknuto tlačítko stop. Avšak né každý krok umožňuje nastavit breakpoit, například pro krok algoritmu, kde je uvedeno jen informativní návěští pro snadnější orientaci uživatele. Celý algoritmus převodu můžete vidět na obrázku (Obr. 10.1).

Nyní bych rád přiložil ještě jeden obrázek (Obr. 12.1) s diagramem, který ukazuje, jak celý algoritmus funguje. Uzly s kolečkem značí jednotlivé kroky algoritmu a šipky s textem napravo znázorňují podmínky, za kterých se přechází z jednoho kroku do druhého. Pokud má šipka u sebe text `always`, tak se následující krok provede vždy. Diagram je dosti názorný, proto jej zde nebudu detailněji popisovat.



Obr. 12.1 Diagram detailně popisující chování algoritmu převodu bezkontextové gramatiky na zásobníkový automat

13 Výstupní widget zásobníkového automatu

Zásobníkový automat, který můžete vidět na obrázku (Obr. 10.1) je zde tvořen žlutými kolečky, uvnitř kterých je pojmenování daného stavu. Pokud je jméno stavu delší, než průměr kolečka, tak se mění tvar na elipsu širokou právě tak, aby obsáhla vnitřní text. Počáteční stav je označen tak, že do něj vede tlustá šipka zleva, která nezačíná v žádném ze stavu zásobníkového automatu. Koncové stavy zásobníkového automatu jsou označeny druhým vnitřním kolečkem.

Přechodová pravidla jsou symbolizovaná šipkou s textem, která lze vést z jakéhokoliv stavu do dalšího a to i z, a do toho stejného. Pokud přechodové pravidlo začíná a končí v tom samém stavu, tak je pak označeno částečně zakrytým kruhem nad stavem, nad kterým je text přechodového pravidla.

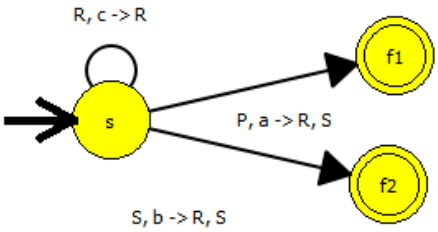
Na obrázku si taktéž můžete všimnout, je vlevo nahoře tohoto widgetu se nachází zelený text `CORRECT`, ten značí že zásobníkový automat je ekvivalentní se správným řešením. Pokud by nebyl, tak by tam byl červený text `WRONG`.

Pro ilustraci zde uvádím příklad zásobníkového automatu $M = (Q, \Sigma, \Gamma, R, q_0, S, F)$ kde:

- $Q \in \{s, f1, f2\}$,
- $\Sigma \in \{a, b, c\}$ je konečná vstupní abeceda,
- $\Gamma \in \{P, R, S\}$ je konečná abeceda zásobníku,
- $R \in \{\{R, s, c \rightarrow s, R\}, \{P, s, a \rightarrow f1, R, S\}, \{S, s, b \rightarrow f2, R, S\}, \}$ je množina, přechodových pravidel zapsaných v šipkové notaci,
- s je počáteční stav z množiny Q,
- S je počáteční symbol na vrcholu zásobníku z množiny Γ,
- $F \in \{f1, f2\}$ je množina koncových stavů, kde $F \subseteq Q$.

viz obrázek (Obr. 13.1).

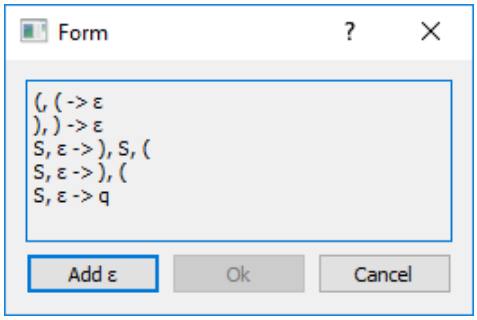
Pro tvorbu zásobníkového automatu slouží kreslící plocha, uvnitř které se v pravém horním rohu nacházejí 4 tlačítka. Pokud je vybráno první tlačítko s ikonou kurzoru myši, pak lze jednotlivé stavy zásobníkového automatu přemisťovat v kreslící ploše. Taktéž lze označovat, jak přechodová pravidla, tak stavy. Pokud máme označen jeden nebo více stavů, nebo přechodových pravidel, lze je smazat jediným klikem na nejpravější tlačítko s ikonou odpadkového koše.



Obr. 13.1 Příklad zásobníkového automatu

Po kliknutí na tlačítko s ikonou kruhu, uvnitř kterého je obrázek plus, se nastaví mód přidávání stavů, ve kterém každým klikem do kreslící scény se na daném místě objeví nový uzel s unikátním jménem. Standardně jsou takto nové stavy označeny vzrůstající sekvencí čísel začínající nulou.

Pokud označíme tlačíko s ikonou šipky, pak můžeme kreslit ve scéně přechodová pravidla. Pravidla se vkládají tak, je se stiskne levé tlačítko myši uvnitř plochy elipsy označující stav automatu, drží se stisklé a uvolní se v jiném nebo stejném stavu. Po uvolnění levého tlačítka myši vyskočí dialog, ve kterém se zadávají přechodová pravidla. Tento dialog můžeme vidět na obrázku (Obr. 13.2).



Obr. 13.2 Okno dialogu pro vkládání přechodová pravidla v zásobníkovém automatu

Pokud vkládáme první pravilo mezi stavy automatu, tak je dialog vyplněn šedým textem informujícím v jakém formátu se jednotlivé pravidla zapisují. Formát je takový, že každé pravidlo začíná symbolem, který se vybírá ze zásobníku, následuje vstupní symbol abecedy oddělený čárkou, následuje šipka ("->") a za ní je seznam symbolů, které se vkládají na zásobník. Záměrně se zde nezapisuje z jakého na jaký stav se má zásobníkový automat přesunout po provedení pravidla, tak aby uživatel nemohl udělat chybu. Tuto informaci zvolil už tím, že vytvořil šipku, která vždy musí v nějakých stavech začínat a končit.

Daný uzel lze označit jako koncový, či počáteční kliknutím pravým tlačítkem uvnitř elipsy a zaškrtnutím příslušného checkboxu. Tuto kontextovou nabídku, kde je vybrán

daný uzel jako počáteční a zároveň jako koncový můžete vidět na obrázku (Obr. 13.3).

Obr. 13.3 Kontextová nabídka pro nastavování počátečního a koncového stavu automatu

14 Widget s aktuálními proměnnými z algoritmického widgetu

V tomto widgetu se zobrazují proměnné použité v algoritmickém widgetu spolu s jejich aktuálními hodnotami. Tyto proměnné se zobrazují pouze tehdy, pokud je vybrán mód krokování algoritmu. Vzhledem k tomu, že jsem tento widget implementoval pomocí ovládacích prvků, které umožňují přijímat vstup ve formátu html, tak jsem mohl použít základní formátování textu, které jsem využil k zvýrazňování a obarvování proměnných tak, aby měly stejnou vizuální podobu s proměnnými použitými v algoritmickém widgetu a to vše proto, abych usnadnil orientaci uživateli v dané konverzi.

III. IMPLEMENTACE

TODO: Korekce implementační části. TEXT TÉTO ČÁSTI JE ZATÍM NAPSÁN POUZE NA HRUBO A POTŘEBUJE KOREKCE.

V této části práce se detailněji podíváme na to jak samotný program vznikal, z jakých částí se skládá, jak jej sestavit ze zdrojových souborů a co všechno jeho programování zahrnovalo.

Aplikaci jsem programoval v jazyce C++ normy C++ 11. A nejen pro tvorbu grafického uživatelského prostředí jsem zvolil multiplatformní framework Qt. Všechny zdrojové soubory jsou publikovány na serveru GitHub.

Prvně si ukážeme jak získat zdrojové soubory a jak je můžeme zkompilovat. Následně si probereme, refaktoring existujícího kódu, který jsem psal v rámci mé bakalářské práce. A poté si probereme nejduležitější stavební prvky programu.

15 Stažení zdrojových souborů a jejich následná kompilace

Všechny zdrojové soubory jsou opensource a jsou veřejně k dispozici na serveru github všetně této práce jak ve formátech pdf a LaTeXna adrese `https://github.com/navrkald/regularConvertor/`. Celou práci jsem verzoval pomocí programu Git a proto si vše můžete naklonovat pomocí příkazu `git clone https://github.com/navrkald/regul` Github umožňuje mimojiné zobrazovat různé statistiky ohledně množství a rozložení commitů. Github měl pro mě taktéž výhodu, že práce byla uložena na veřejně přístupném serveru a já tak mohl vyvíjet střídavě z různých míst a zařízení. Dále pak jsem mohl mít jistotu, že Github má vše zálohované a i kdyby ne, tak samotný Git má tu úžasnou vlastnost, že každým naklonování mého repositáře vzniká kompletní off-line záloha a vzhledem k faktu, že jsem pracoval z více zařízení, tak mám několik záloh.

Opensource řešení jsem zvolil jednak z důvodů toho, že jsem chtěl, aby moje zdrojové soubory mohl vidět celý svět, aby je kdokoliv mohl upravit a pomocí *pull requestu*, které mám ve ve svém repositáři povolené, dále rozvíjet tuto skvělou aplikaci. Druhým a to právním důvodem bylo, že jelikož nepoužívám multiplatformí Qt framework v komerční licenci, za kterou bych musel platit, jsem povinen všechny své zdrojové soubory zveřejnit.

V předchozích odstavcích jsme se dozvěděli, jak získat zdrojové soubory, nyní si už jen zbývá ukázat, jak zkompilovat zdrojové soubory do spustitelného souboru.

Základní dvě možnosti jsou, použít nativní integrované vývojové prostředí QtCreator a nebo použít Qt plugin do Visual studia. Obě tyto možnosti začínají tím, že si musíme stáhnou samotný Qt framework pro danou platformu, na které zrovna vyvíjíte. Zde je odkaz na všechny možné instalátory Qt opensource frameworku, pro všechny hlavní platformy: `https://www.qt.io/download-open-source/#section-2`. Pro windows jsou k dispozici mimo jiné verze pro kompilátor visual studia a MinGW. Já jsem použil kompilátor Qt 5.8.0 for Windows 64-bit (VS 2015, 1.0 GB). Součásti balíčku je i integrované vývojové prostředí QtCreator spolu se zdrojovými kódy a binární podobou Qt frameworku.

Zdrojové soubory jsou užitečné ve chvíli, pokud chcete udělat staticky zkompilovanou aplikaci používající Qt framework, tak aby v sobě obsahovala všechny Qt závyslosti. Výhodou je, že pak můžete distribuovat vaši aplikaci v takzvaném "portable"formátu, kde aplikace neobsahuje instalátor, nevýhodou je za vyšší velikost výsledného binárního souboru. Jelikož je statická kompilace nad rámec této práce, pouze odkáži na odkaz na internetu: `https://wiki.qt.io/Building_a_static_Qt_for_Windows_using_MinGW`.

Po stažení a naistalování Qt frameworku včetně Qt Creatoru a zdrojových souborů, které se hodí i v případě nejasností v dokumentaci Qt, můžete již vyvýjet, tak že si

v něm otevřete projektový soubor, který se nachází v cestě <cesta k naklonovanému repositáři>/src/RegularConvertor.pro. Pak již stačí projekt sestavit pomocí `Build -> Build All` a pak spustit `Build -> Run`. Pro debuging programu pouze odkážu na nastavení debugeru na url: `http://doc.qt.io/qtcreator/creator-debugger-engines.html`.

Mě osobně po čase přestávalo prostředí QtCreatoru stačit a potřeboval jsem více pokročilé IDE, proto jsem si nainstaloval do Visual Studia plugin QtPackage dostupný online: `https://marketplace.visualstudio.com/items?itemName=havendv.QtPackage`. Plugin v sobě neobsahuje celý Qt framework, proto jej je nutné předtím stáhnout. Po nastavení pluginu už pak jen stačí ve Visual Studiu spustit a zbuildit pomocí klávesové zkratky `F5`.

16 Refaktor existujícího kódu

Refaktoring je činnost při, které se zlepšuje kvalita kódu bez přímého vlivu na uživatele aplikace.

Nerefaktorovaný kód vede k tomu, že jen obtížně mohou programátoři rozumět kódu a to vede k častým chybám a prodloužením vývoje.

Všechny zdrojové soubory nutné pro úspěšnou kompilaci kódu jsem přesunul do samostatné složky standardně pojmenované *src*, podle anglického slova *sources*.

Kód by měl být pokud možno homogenní a měl by v něm být dodržován systém pojmenovávaní.

16.1 Konvence pojmenování částí kódu

Pro pojmenovávaní proměnných, tříd a typů jsem použil systém používaný ve firmě AVG. Vše popisuje následující tabulka.

Tab. 16.1 Konvence pro pojmenovávání tříd a proměnných

konvence	poznámka
m_<název členské proměnné>	Písmeno *m* podle anglického slova member.
C_<Název třídy>	Písmeno *C* Podle anglického slova Class.
I_<Název rozhraní>	Písmeno *I* podle anglického slova Interface.

Dalšími konvencemi jsou pojmenovávání pomocí "CammelCase". To znamená, že pokud se nějaký název skládá z více slov, tak jsou jednotlivá slova od sebe graficky oddělena pomocí toho, že každé další slovo je od předchozího odděleno velkým písmenem.

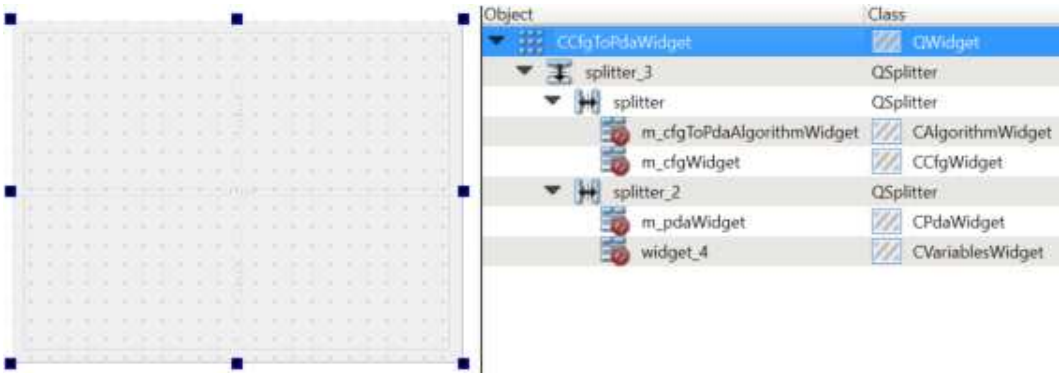
Při pojmenovávání souborů jsem se snažil dodržet konvenci, že název souboru se skládá pouze z malých písmen následovaný příponou souboru.

Jelikož množství a rozsah zdrojových souborů je značný, pojmenovávací konvence popsané výše jsem použil jen u nově psaných nebo starých upravovaných částí.

16.2 Přepsání třídy hlavního okna

Příliš velké metody nebo třídy nejsou tolik přehledné a udržovatelné. Proto jsem se rozhodl přepsat moji třídu hlavního okna do více modulů, chcete-li tříd.

V mém původním návrhu bakalářské práce jsem plně neovládal techniku zvanou *Promoting*. Je to technika v Qt Designeru, kde si navrhnu pro dané okno navrhnu rozložení widgetu a následně nastavím, že místo originálních widgetů se mají použít mnou upravené odděděné. Toto je možné, protože každý grafický prvek má za předka QWidget.



Obr. 16.1 Příklad promoting na widgetu pro konverzi bezkontextové gramatiky na zásobníkový automat.

Na obrázku (Obr. 16.1) můžete vidět rozložení widgetů pro konverzi bezkontextové gramatiky na zásobníkový automat. Co nemůžete vidět je jinde navrhlá konkrétní grafická podoba. Tato konverze se skládá ze 4 widgetů (od levého horního rohu k pravému dolnímu rohu): widget pro bezkontextovou gramatiku, widget vizualizující konverzi, widget pro zásobníkový automat a vpravo dole widget, kde se ukazují aktuální hodnoty proměnných použitých v konverzním widgetu.

Původně vytváření grafického rozvržení widgetů jsem měl pro všechny konverze implementované v třídě hlavního okna. Naopak nyní jsou všechny konverze popsány graficky v souborech s koncovkou **.ui* a logika konverzí v asociovaných zdrojových souborech s koncovkami **.cpp* a **.h*. Toto jsem dělal ve jménu jedné z hlavních myšlenek OOP a to zapouzdření.

Jelikož takto přepsané všechny widgety popisující jednotlivé konverze obsahují společnou množinu vlastností, tak všechny dědí od společného rozhraní ICentralConversionWidget, které má čistě virtuální metody ConnectChangeMode() ConnectStatusMessage(). První metoda má za úkol propojit notifikaci o změně módu z hlavního okna do tříd které zaštiťují jednotlivé konverze. A druhá naopak propojuje notifikace o různých případných chybových stavech z konverzního widgetu do hlavního okna, kde se pak tyto notifikace zobrazují ve statusové liště. Tento princip, kdy jednotlivé třídy mají společné vlastnosti, které zaštiťuje rozhraní a kde se pak v programu pracuje s ukazatelem na rozhraní se nazívá polimorfizmus.

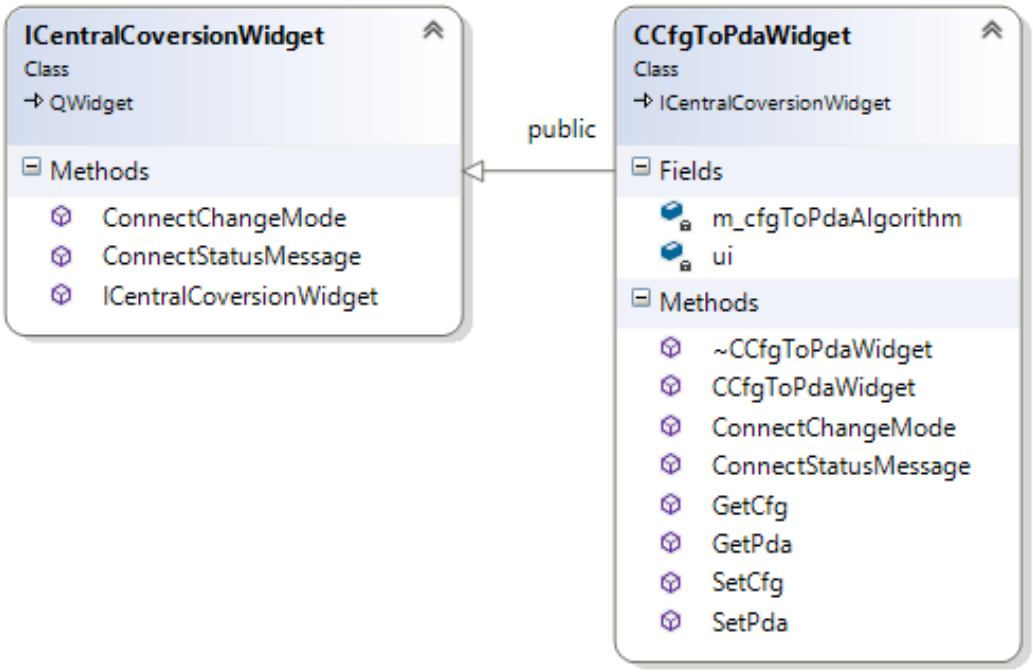
Pro vytváření centrálního widgetu, ve kterém se zobrazují jednotlivé konverze složí ve třídě `MainWindow` metoda `PrepareConversionWidget(Conversions conversion)`, která má jeden vstupní parametr informující o tom, která konverze se má nastavit.

Vzhledem k tomu, že každá z konverzí má jiná vstupní a výstupní data, tak princip polymorfizmu nešel použít a bylo potřeba přetypovat na skutečný typ třídy v případech, kdy bylo potřeba uložit, nebo načít konverzi ze souboru, a stejně tak, pokud se načítala

konverze v příkladů zabudovaných v aplikaci.

16.3 Popis nejduležitějších tříd zajišťující převod bezkontextové gramatiky na zásobníkový automat

Nyní si pojďme ukázat na diagramu tříd, z čeho se zkládá třída `CCfgToPdaWidget` zaštiťující převod bezkontextové gramatiky na zásobníkový automat. Tento diagram můžeme vidět na obrázku (Obr. 16.2).



Obr. 16.2 Diagram tříd třídy CCfgToPdaWidget

Pro widgety u kterých se mění popisek jsem přidal metodu *SetCaption()*, která jim nastaví správný text.

TODO: Diagram tříd pro vICentralCoversionWidget a popis method

17 Drobné vylepšení existujícího kódu

Pro uložení existujících konverzí do souboru jsem přidal klávesovou zkratku `CTRL+S` podle anglického *Save* a pro uložení `CTRL+L`, podle anglického *Load*.

18 Analýza nástrojů vytváření vizualizací z existujícího kódu

Při hledání nástrojů pro vizualizaci a dokumentaci existujícího kódu jsem měl následují požadavky:

* Všechny nástroje musejí být bezplatné. * Musejí uměl vizualizovat diagram tříd a propojení tříd pomocí Qt mechanizmu Signálů a slotů. * Musí umět ideálně vygenerovat požadované vizualizace automaticky, tak aby se při změně kódu automaticky aktualizovali a nemuseli se vytvářet pokaždé ručně.

18.1 Qt ModelEditor

Prvním nástrojem je *ModelEditor*. Tento nástroj jsem zkusil jako první protože je ve formě pluginu do IDE *QtCreator*, který jsem používal pro programování této diplomové práce. *ModelEditor* umožňuje dělat sice jednoduché, ale přehledné UML diagramy. Zásadní nedostatek, je že umožňuje například diagram tříd generovat jenom částečně a to tak, že se přetáhne zdrojový soubor do modelovacího editoru, kde se vytvoří element s názvem třídy, ale metody a členské se musí dopisovat ručně. Na domovské stránce (`https://wiki.qt.io/ModelEditor`) tohoto nástroje jsem se nikde nedočetl, že by se na této funkcionalitě mělo někdy pracovat, proto jsem tento nástroj prozatím zavrhl.

Another link to Qt ModelEditor `http://doc.qt.io/qtcreator/creator-modeling.html`

18.2 Doxygen a Graphviz

Druhým nástrojem je *Doxygen*, který je určen na automatickou tvorbu dokumentace. Tento nástroj byl ve svém počátku určen právě pro C++/Qt. *Doxygen* pro tvorbu pokročilých grafů potřebuje doinstalovat *Graphviz* a nastavit systémovou proměnnou PATH na cestu ke spustitelným souborům *Graphviz*. Funguje to tak, že *Doxygen* vytvoří textový popis grafu a *Graphviz* pomocí spustitelného programu *Dot* umí vytvořit grafickou podobu jak ve vektorových, tak v rastrových formátech. Aby bylo generování co nejjednodušší vytvořil jsem si konfigurační soubor pojmenovaný "Doxyfile"v adresáři "src". Následně se celá dokumentace vygeneruje automaticky pomocí zavolání příkazu doxygen ve stejném adresáři ve kterém se nachází "Doxyfile". *Doxygen* umí vytvořit UML diagram tříd. U každé třídy ukáže od kterých tříd dědí a také názvy proměnných a metod spolu s jejich modifikátorem viditelnosti (public, protected, private). Zásadním problémem je, že neukazuje názvy typů členských proměnných, ani návratové typy a typy parametrů metod. Pro automatické generování dokumentace z kódu je tento nástroj ideální, ale já chtěl v této diplomové práci mít kompletní diagram tříd včetně typů a návratových hodnot proto jsem zkoušel ještě třetí nástroj.

18.3 Diagram tříd ve Visual studio Express 2015

Třetím nástrojem, který uspokojil všechny požadavky na diagram tříd se nacházel v IDE *Visual Studio* ovšem s jednou malou komplikací a to, že si neporaďil plně s parsováním tříd, které mají definované makro Q_OBJECT a pro správnou funkčnost bylo nutné toto makro dočasně zakomentovat.

Diagram konkrétní třídy se ná vygenerovat tak, že se pravým tlačítkem klikne na hlavičkový soubor třídy a následně vybere možnost "zobrazit diagram třídy". Po kliku kamkoliv do volného místa se dá vyvolat nabídka na export do různých formátů. Dá se i zvolit jestli chceme zobrazit celou signaturu, tj. návratové typy, jména a typy parametrů a typy členských proměnných. Jediná drobná výtka by mohla být, že modifikátor viditelnosti je a zda se jedná o metodu nebo proměnnou je značen z hlediska UML nestandardní ikonou. Zásadní je, že diagram tříd vygenerovaný tímto nástrojem obsahuje všechnu podstatné informace, proto jej použiji pro generovaní obrázků tříd do této diplomové práce.

TODO: Implementace hlavního okna, zmínit se třídě MyTooltipQMenu, která implementuje ukazování obrázků jako tooltipů nad položkami v menu a tím že menu Context-Free grammar je povýšena na MyTooltipQMenu a uvest diagram třídy. Zmínit se, že kazdy example má u sebe screenshot

TODO: Když bude velká nouze, tak popsat jak jsem si konfigurovat VS (F4, barvičky...). Že v Qt creatoru jsem používal tabulátory místo mezer, jak jsou defaultně nastavené.

TODO: Napsat proč jsem nepoužil iteratory, ale listy v CFG to PDA (kvuli moznosti nacitani a ukladani jednotlivych stepů.)

18.4 Simple tool to visualize connections between signals and slots in Qt

Pro vizualizaci propojení pomocí Qt slotů a signálů jsem akorát našel jednoduchý nástroj, jehož zdrojový text je uveden na adrese `http://hackatool.blogspot.cz/2013/05/simple-tool-to-visualize-connections.html`. Tento nástroj pracuje na principu prohledání zdrojových souborů na základě regulárního výrazu a následně vygeneruje textový popis grafu propojení ve formátu, kterému rozumí program Graphviz zmíněný výše.

ZÁVĚR

TODO: Ujasnit si o čem chci v diplomce psat

TODO: Do závěru napsat počet normostran a metriky kody, kolik radku, znaku, kolik zmen jsem udelal v kodech od diplomky...

TODO: Zhodnoceni, co se všechno povedlo. (Ne moc kritiky!!!)

V této práci jsem se snažil k hůře stravitelné teorii přikládat množství příkladů, aby byla tato problematika pochopitelná a přínosná pro studenta, který by si tuto práci chtěl půjčit v univerzitní knihovně. Z vlastní zkušenosti vím, že pro lepší pochopení dané problematiky je kromě suché teorie nutné přiložit i příklady. Dával jsem je zde i pro to, že konečným cílem diplomové práce bude kromě textu nutné vytvořit i funkční aplikaci, která má sloužit především studentům, ale samozřejmě ji mohou využít i profesoři na naší škole k prezentaci přednášené látky pro lepší přiblížení studentům. Chtěl jsem, aby i student nepříliš seznámený s danou problematikou mohl po přečtení této práce začít používat mou aplikaci bez obtíží.

SEZNAM POUŽITÉ LITERATURY

[1] KOPKA, Helmut, Patrick W DALY a Jan GREGOR. *Latex: podrobný průvodce.* Vyd. 1. Brno: Computer Press, 2004, 576 s. ISBN 80-7226-973-9.

[2] *Chomsky hierarchy* [online]. [cit. 2015-11-16]. Dostupný z WWW: `https://en.wikipedia.org/wiki/Chomsky_hierarchy`.

[3] *Context-free language* [online]. [cit. 2015-11-17]. Dostupný z WWW: `https://en.m.wikipedia.org/wiki/Context-free_language`.

[4] *Pushdown automaton* [online]. [cit. 2015-11-17]. Dostupný z WWW: `https://en.wikipedia.org/wiki/Pushdown_automaton`.

[5] *Context-free grammar* [online]. [cit. 2015-11-17]. Dostupný z WWW: `https://en.m.wikipedia.org/wiki/Context-free_grammar`.

[6] *Backus-Naur Form* [online]. [cit. 2015-11-18]. Dostupný z WWW: `https://en.wikipedia.org/wiki/Backus-Naur_Form`.

[7] HOPCROFT, John E, Rajeev MOTWANI a Jeffrey D ULLMAN. *Introduction to automata theory, languages, and computation.* 2nd ed. Boston: Addison-Wesley, 2001, xiv, 521 s. ISBN 0201441241.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

CFG	Context-free grammar, česky taktéž bezkontextová gramatika
PDA	Pushdown automaton, česky též zásobníkový automat
BNF	Backusova-Naurova forma. (Notace pro zápis bezkontextové gramatiky.)

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I. NÁZEV PŘÍLOHY

Obsah přílohy