VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Doplnění editoru a generátoru grafů konečných automatů

Addition of New Features to a Generator of Finite-automata Graphs

2011 Pavel Byma

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 6. května 2011 .....

Rád bych na tomto místě poděkoval vedoucímu bakalářské práce Ing. Martinu Kotovi, Ph.D. za užitečnou metodickou pomoc a rady při zpracování bakalářské práce.

Abstrakt

Bakalářská práce navazuje na práci „Editor a generátor grafů konečných automatů“, kterou vytvořil Petr Hrubý, určenou pro práci s grafickým balíkem GasTeX používaným v LaTeXu. Funkce pro tvorbu a úpravu konečných automatů, které již v editoru existovaly byly zanechány a doplněny o překladač příkazů GasTeXu, který byl zpracován v JavaCC. Do editoru byly doplněny algoritmy, které podporují mnohem větší spektrum grafických úprav pro kreslení vrcholů a hran grafů konečných automatů. Do programu bylo přidáno nastavení globálních parametrů, které ovlivňují vzhled vykreslování všech následně kreslících objektů.

Klíčová slova: konečný automat, editor, LaTeX, GasTeX, Java, JavaCC, překladač, bezkontextová gramatika

Abstract

Bachelor thesis continues the work of „Editor and Generator of Finite Graph Automata“, created by Petr Hrubý, intended to work with the graphics package used in GasTeX LaTeX. Functions to create and edit finite automata, which already existed in the editor was left and added a command interpreter GasTeX, which was prepared in JavaCC. The editor has been added algorithms, which supports a much wider range of graphic editing for drawing graphs of vertices and edges of finite automata. The program was added to set the global parameters that affect the appearance of rendering all subsequent drawing objects.

Keywords: Finite-automata, Editor, LaTeX, GasTeX, Java, JavaCC, Compiler program, Context-free grammar

Seznam použitých zkratek a symbolů

GasTeX	–	Graphs and Automata Simplified in TeX
JavaCC	–	Java Compiler Compiler
XML	–	eXtensible Markup Language
DTD	–	Document Type Definition
DOM	–	Document Object Model

Obsah

1 Úvod **5**

2 Konečné automaty **6**
2.1 Konečné detemirnistické automaty 6
2.2 Konečné nedetemirnistické automaty 7

3 GasTeX **9**
3.1 Parametry GasTeXu 9
3.2 Příkazy GasTeXu 12

4 Analýza původního programu **15**
4.1 Implementace 15
4.2 Grafické rozhraní 16
4.3 Vykreslování objektů 16
4.4 Funkce pro tvorbu konečných automatů 17
4.5 Vstupní a výstupní funkce 18
4.6 Zhodnocení analýzy 19

5 Překladač GasTeX kódu **20**
5.1 Implementace překladače v JavaCC 20
5.2 Sestavování gramatických pravidel 20
5.3 Zpracování tokenů 22
5.4 Generované soubory 24

6 Úprava původního programu **26**
6.1 Propojení překladače s původním programem 26
6.2 Propojení paměťových struktur 28
6.3 Grafické zobrazení 28
6.4 Parametry 30
6.5 Funkce s automaty 31
6.6 Vstupní a výstupní data 31

7 Závěr **33**

8 Reference **34**

Přílohy **34**

A Příloha **35**
A.1 Program 35
A.2 Uživatelský manuál 35
A.3 Programátorská dokumentace 35
A.4 Vlastní text bakalářské práce 35

Seznam tabulek

1 Ukázka tabulky přechodů 7

Seznam obrázků

1 Ukázka deterministického konečného automatu 6
2 Ukázka nedeterministického konečného automatu 7
3 UML schéma základu programu 15
4 Hlavní okno programu 26
5 Okno pro načtení příkazů GasTeXu 27

Seznam výpisů zdrojového kódu

1 Definování hlavičky pro LaTeX v původním programu 19
2 Ukázka příkazu *node* pro vykreslení vrcholu v GasTeXu 21
3 Ukázka definovaných tokenů 21
4 Funkce pro definování parametrů 23
5 Hlavní funkce překladače 24
6 Nastavování překladače JavaCC 25
7 Zpracování překládaného textu 27
8 Defaultní nastavení globálních parametrů 30
9 Část souboru DTD 32

1 Úvod

Bakalářská práce „Doplnění editoru a generátoru grafů konečných automatů“ navazuje na bakalářskou práci Petra Hrubého „Editor a generátor grafů konečných automatů“. Ta se zabývá vytvořením grafického uživatelského rozhraní pro vytváření grafů konečných automatů a jejich export do GasTeXu. Mojim úkolem bylo doplnit editor o překladač GasTeXového kódu a další funkce rozšiřující program.

Jediný program, který jsem našel a zabývá se danou tématikou je JasTeX [7]. Tento program dokáže kreslit konečné automaty a generovat jejich GasTeXový kód. Nedokáže ale vytvářet automaty automaticky podle zadaných kritérií například přechodovou tabulkou. Také neumí načítat GasTeXový kód.

Konečný automat je výpočetní model používaný v informatice pro formální jazyky. Je to systém, který nabývá konečného počtu vnitřních stavů, které se mění na základě vnějších podnětů přicházejících do systému. Výstupem systému je informace, zda se automat nachází v přijímacím nebo nepřijímacím stavu. Více se můžete o konečných automatech dovědět v kapitole 2.

Pro kreslení konečných automatů v LaTeX se využívá grafického balíku zvaného GasTeX. Tato knihovna obsahuje sadu maker pro snadné kreslení obrázků grafů automatu. V kapitole 3 se můžete dovědět, jaká makra se pro kreslení grafů konečných automatů používají a jak mohou ovlivnit výsledný vzhled jejich parametry.

Než se pustíme do samotného úkolu bakalářské práce, je nutné zanalyzovat původní projekt a zhodnotit, které funkce by bylo dobré zachovat a využít pro náš úkol. Proto byl vytvořen popis původního programu, který je v kapitole 4.

Po analýze bylo možno přistoupit k samotnému úkolu (kapitola 5). Ten byl vytvořit překladač pro interpretaci GasTeXových příkazů, který by byl doplněn do původního editoru. Překladač byl vytvořen v JavaCC. Jedná se o program, který dokáže vytvářet podle zadaných pravidel pro lexikální a syntaktickou analýzu překladače zadaného jazyka. Jeho výstupem jsou soubory, které jdou přeložit běžným překladačem pro Javu a tím je můžeme snadno integrovat do původního programu.

Z původního programu nezůstal kámen na kameni a o tom, co všechno bylo nakonec změněno, se můžete dočíst v kapitole 6. Mezi hlavní změny patří grafické upravení vykreslování vrcholů a hran konečného automatu nebo nastavování globálních parametrů pro vykreslované příkazy GasTeXu.

2 Konečné automaty

2.1 Konečné detemirnistické automaty

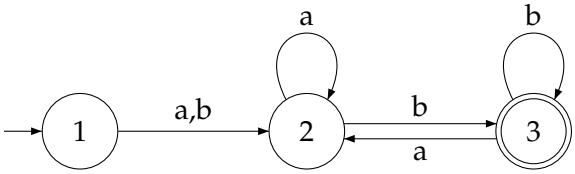
Konečný automat je výpočetní model používaný v informatice pro formální jazyky. Může nabývat konečného počtu vnitřních stavů, které se mohou měnit na základě vnějších podnětů. Pro daný stav a daný podnět je jednoznačně určeno, který stav automatu bude následující. Výstupem automatu je dvouhodnotová informace, která udává, je-li právě automat v přijímacím či nepřijímacím stavu.

Definice konečného automatu 2.1 [1].

Definice 2.1 *Konečný automat je uspořádaná pětice $A = (Q, \Sigma, \delta, q_0, F)$, kde*

- *Q je konečná neprázdná množina stavů,*
- *Σ je konečná neprázdná množina zvaná (vstupní) abeceda,*
- *$\delta : Q \times \Sigma \to Q$ je přechodová funkce,*
- *$q_0 \in Q$ je počáteční (iniciální) stav,*
- *a $F \subseteq Q$ je neprázdná množina přijímajících (koncových) stavů.*

Pro způsob vizualizace konečného automatu se používá orientovaného grafu, jehož vrcholy znázorňují vnitřní stavy automatu a ohodnocené hrany (šipky) mezi nimi jsou přechody pro jednotlivé vstupní podněty, které reprezentují symboly zvolené abecedy. Duplicitní přechody spojíme do jednoho a sepíšeme k němu všechny náležící symboly abecedy odstraněných přechodů. Na obrázku 1 můžeme vidět příklad konečného automatu.



Obrázek 1: Ukázka deterministického konečného automatu

Ke značení zápisu konečných automatů se často využívá tabulky přechodové funkce. Tabulka má řádky označeny stavy automatu a sloupce symboly abecedy. Jednotlivá políčka pak představují přechodovou funkci δ (ze stavu se pomocí znaku abecedy přejde do stavu v políčku). Šipky vlevo a vpravo určují počáteční a koncový stav. V tabulce číslo 1 jsou zaznamenány přechodové funkce ukázkového konečného automatu z obrázku 1.

	δ	a	b
$\rightarrow$	1	2	2
	2	2	3
$\leftarrow$	3	2	3

Tabulka 1: Ukázka tabulky přechodů

2.2 Konečné nedetemirnistické automaty

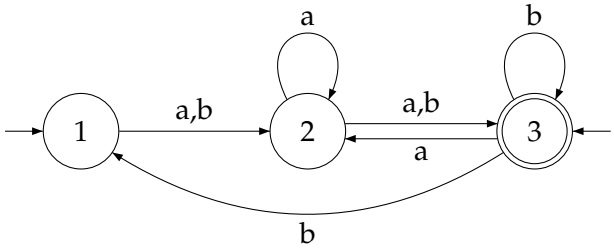
Nedeterministické konečné automaty mají možnost oproti deterministickým přejít na zadaný znak abecedy do více stavů nebo také můžeme přecházet po hranách značených symbolem ε. Tyto tzv. ε-přechody nevyžadují žádný vstupní znak, ale můžeme jimi přejít přímo. Nedeterministické automaty mají rovněž možnost obsahovat více vstupních stavů. Slovo zadané abecedy bude přijato, pokud alespoň jeden možný výpočet průchodu automatu skončí v přijímacím stavu.

Definice nedeterministického konečného automatu 2.2 [1].

Definice 2.2 *Nedeterministický konečný automat je uspořádaná pětice $A = (Q, \Sigma, \delta, I, F)$, kde*

- *Q je konečná neprázdná množina stavů,*
- *Σ je konečná neprázdná množina zvaná (vstupní) abeceda,*
- *$\delta : Q \times (\Sigma \bigcup \{\epsilon\}) \rightarrow 2^Q$ je (nedeterministická) přechodová funkce ,*
- *$I \in Q$ je neprázdná množina počátečních stavů,*
- *a $F \subseteq Q$ je neprázdná množina přijímajících (koncových) stavů.*

Na obrázku 2 máme nedeterministický konečný automat se dvěma počátečními stavy 1 a 3 a abecedou {a,b}. Zatímco slovo $'bba'$ při vstupu do počátečního stavu 1 nebude přijato, při vstupu do počátečního stavu 3 se dostane do přijímacího stavu a tudíž bude přijato i celým nedeterministickým automatem.



Obrázek 2: Ukázka nedeterministického konečného automatu

Každý nedeterministický automat se dá převést na ekvivalentní deterministický automat. Ne vždy ale mají takové automaty rozumnou velikost, může se jednat až o 2^n vrcholů oproti n stavům nedeterministického automatu.

3 GasTeX

GasTeX [2] je grafický balík, který obsahuje sadu příkazů (kapitola 3.2) a jejich parametrů (kapitola 3.1) pro vykreslování vrcholů, hran a dalších objektů. Rozšiřuje balík maker pro snadné kreslení obrázků grafů, automatů, sítí, schémat apod. v prostředí LaTeXu. LaTeX je balík maker programu TeX, který dovoluje autorům sázet text a tisknout svá díla ve velmi vysoké typografické kvalitě. Pro překlad LaTeXu existuje několik kompilátorů, já jsem pro testování použil software MiKTeX v aktuální verzi 2.9. GasTeX produkuje postskriptový kód, který neumí zpracovávat ***pdflatex*** (překladač LaTeXu do pdf). Pro vytvoření pdf je nutné přeložit postskriptový soubor pomocí ***ps2pdf***. Grafický balík GasTeX vytvořil Paul Gastin, který je profesorem na francouzské univerzitě ENS v Cachan.

3.1 Parametry GasTeXu

Parametry GasTeXu definují vlastnosti objektů a tím i jejich vzhled při vykreslení, jako jsou například velikost objektu, tloušťka nebo zakřivení čar, vzdálenost textového popisu apod. Parametry se dají nastavit pouze pro jeden objekt anebo pomocí příkazu *gasset* pro všechny používané objekty.

- **Nw** = šířka oválu vrcholu
- **Nh** = výška oválu vrcholu
- **Nmr** = maximální poloměr rohu vrcholu, použije se minimum z hodnot Nw/2, Nh/2 nebo Nmr, vrchol s Nmr = 0 bude obdélník (čtverec)
- **Nadjust** = přizpůsobení vrcholu velikosti obsahu textu, lze použít i kombinaci znaků
  - *w* = přizpůsobení šířky vrcholu
  - *h* = přizpůsobení výšky vrcholu
  - *n* = nepřizpůsobovat
- **Nadjustdist** = vzdálenost mezi textem a okrajem vrcholu při přizpůsobení velikosti pomocí Nadjust
- **Nframe** = nastavuje zobrazení ohraničení vrcholu
  - *y* = zobrazit ohraničení
  - *n* = nezobrazovat ohraničení
- **Nfill** = nastavuje zobrazení pozadí vrcholu, automaticky nastaveno při použití parametrů *fillgray* a *fillcolor*
  - *y* = zobrazit pozadí
  - *n* = nezobrazovat pozadí

- **ExtNL** = nastavuje zobrazení textového popisku uvnitř nebo vně vrcholu
  - *y* = popisek vně vrcholu
  - *n* = popisek uvnitř vrcholu
- **NLangle** = úhel (ve stupních) zobrazení popisku kolem středu vrcholu
- **NLdist** = vzdálenost mezi středem vrcholu a popiskem (ExtNL=n), mezi ohraničením vrcholu a popiskem (ExtNL=y)
- **Nmarks** = kombinace *i, f, r, n* určuje označení vrcholu
  - *i* = označí vrchol šipkou dovnitř
  - *f* = označí vrchol šipkou ven
  - *r* = označí vrchol opakovaným ohraničením
  - *n* = vrchol bez označení
- **ilength** = velikost šipky dovnitř vrcholu
- **iangle** = úhel (ve stupních) natočení šipky dovnitř vrcholu
- **flength** = velikost šipky ven z vrcholu
- **fangle** = úhel (ve stupních) natočení šipky ven z vrcholu
- **rdist** = vzdálenost opakované hrany od ohraničení vrcholu, kladné uvnitř vrcholu, záporné venku
- **sxo** = horizontální offset počátečního bodu křivky
- **syo** = vertikální offset počátečního bodu křivky
- **exo** = horizontální offset konečného bodu křivky
- **eyo** = vertikální offset konečného bodu křivky
- **curvedepth** = určuje hloubku zakřivení křivky definující mezi středem křivky a středem úsečky mezi jejími vrcholy, kladná hodnota zakřivuje vlevo, záporná hodnota vpravo
- **loopwidth** = šířka smyčky
- **loopheight** = výška smyčky
- **loopdiam** = průměr (šířka i výška) smyčky
- **loopangle** = úhel směru smyčky (ve stupních)
- **loopCW** = směr smyčky

  - *y* = ve směru hodinových ručiček
  - *n* = proti směru hodinových ručiček

- **AHnb** = počet šipek na hraně ve směru jejího vytvoření
- **AHdist** = vzdálenost dvou šipek na hraně ve směru jejího vytvoření
- **AHangle** = úhel (ve stupních) mezi hranou ve směru jejího vytvoření a stranou šipky
- **AHLength** = délka strany šipky, která je ve směru vytvoření hrany
- **AHlength** = délka středové osy šipky směřující ve směru vytvoření hrany násobená cosinem úhlu trojúhelníkové šipky
- **ATnb** = počet šipek na hraně proti směru jejího vytvoření
- **ATdist** = vzdálenost dvou šipek na hraně proti směru jejího vytvoření
- **ATangle** = úhel (ve stupních) mezi hranou proti směru jejího vytvoření a stranou šipky
- **ATLength** = délka strany šipky, která je proti směru vytvoření hrany
- **ATlength** = délka středové osy šipky směřující proti směru vytvoření hrany násobená cosinem úhlu trojúhelníkové šipky
- **ELside** = určuje stranu křivky, na které bude popisek
  - *l* = levá strana křivky
  - *r* = pravá strana křivky
- **ELpos** = pozice popisku vůči délce křivky (0..100)
  - *0* = počátek křivky
  - *50* = střed křivky
  - *100* = konec křivky
- **ELdist** = vzdálenost mezi popiskem a křivkou
- **ELdistC** = vzdálenost popisku od křivky se počítá mezi středem popisku a křivkou nebo stranou popisku a křivkou
  - *y* = mezi středem popisku a křivkou
  - *n* = mezi stranou popisku a křivkou
- **linegray** = desetinné číslo mezi 0 a 1, které určuje stupeň šedosti čáry
  - *0* = černá

  - *1* = bílá

- **fillgray** = desetinné číslo mezi 0 a 1, které určuje stupeň šedosti pozadí vrcholu
  - *0* = černá
  - *1* = bílá
- **linecolor** = název barvy určující barvu čáry, je potřeba includovat
  `\usepackage[usenames]{color}`
- **fillcolor** = název barvy určující barvu pozadí vrcholu, je potřeba includovat
  `\usepackage[usenames]{color}`
- **linewidth** = šířka čáry
- **dash** = nastavuje plynulost (přerušovanost) čáry, skládá se ze seznamu opakujících se délek pomlček a mezer, dále se skládá z offsetu, který určuje počátek začátek přerušované čáry od jejího počátku
  - *dash*={}{*0*} = plná čára
  - *dash*={*1.5*}*0* = přerušovaná čára s délkou 1.5 pomlčky a 1.5 mezery
  - *dash*={*0.2 0.5*}*0* = tečkovaná čára, 0.2 pomlčka a 0.5 mezera
  - *dash*={*4 1 1 1*}*0* = přerušovaná řára, střídá se delší a kratší pomlčka
  - *dash=1.51.5* = přerušovaná čára, 1.5 pomlčka a 1.5 mezera, začíná mezerou
  - *dash*={*4*}{*2*} = přerušovaná čára, začátek v polovině první pomlčky
- **arcradius** = poloměr zaoblení vrcholů polygonů, 0 = ostrý úhel
- **polyangle** = úhel (ve stupních) otočení prvního vrcholu polygonu vůči ose x

3.2 Příkazy GasTeXu

Příkazy GasTeXu jsou makra, která umožňují vykreslit jednoduché základní geometrické tvary a křivky. Vyznačují se svým typem, který určuje druh příkazu, seznamem parametrů (viz. předchozí podkapitola 3.1), které ovlivňují vlastnosti objektů a které mohou a nemusí být implementovány a v případě, že implementovány nejsou, se použijí globálně nastavené parametry. Další částí příkazů jsou jejich vlastní atributy mezi nimiž mohou být například název příkazu, x-ové a y-ové souřadnice, názvy jiných příkazů, koordináty řídících bodů křivek nebo třeba vlastní textové popisky, které se budou vykreslovat.

- **gasset**{**parameter=value,parameter=value,...**} = speciální příkaz, který jako jediný nic nevykresluje a trochu se vymyká předešlé definici. Seznam parametrů a vlastních atributů je zde totožný a globálně přenastavuje všechny zadané parametry pro využití ostatními příkazy.

- **rpnode[parameter=value,...](NodeName)(x,y)(n,r)**{**NodeLabel**} = vrchol, který je pravidelným n-úhelníkem
- **node[parameter=value,...](NodeName)(x,y)**{**NodeLabel**} = vrchol, který má tvar oválu
- **imark[parameter=value,...](NodeName)** = značka vrcholu (šipka dovnitř), lze ji nahradit parametrem *Nmarks=i*, ale oproti parametru toto makro umožňuje vykreslit k jednomu vrcholu několik šipek nezávisle na sobě
- **fmark[parameter=value,...](NodeName)** = značka vrcholu (šipka ven), lze ji nahradit parametrem *Nmarks=f*, ale oproti parametru toto makro umožňuje vykreslit k jednomu vrcholu několik šipek nezávisle na sobě
- **rmark[parameter=value,...](NodeName)** = značka vrcholu (opakující ohraničení), lze ji nahradit parametrem *Nmarks=r*, ale oproti parametru toto makro umožňuje vykreslit k jednomu vrcholu několik ohraničení nezávisle na sobě
- **nodelabel[parameter=value,...](NodeName)**{**NodeLabel**} = umožňuje vůči jednomu vrcholu vytvořit několik na sobě nezávislých popisků
- **drawedge[parameter=value,...](startingNode,endingNode)**{**label**} = hrana mezi dvěma vrcholy
- **drawqbedge[parameter=value,...](startingNode,x,y,endingNode)**{**label**} = hrana mezi dvěma vrcholy pomocí kvadratické Bezierovy křivky, první a poslední bod určují vrcholy, prostřední bod určují souřadnice Kartézské soustavy
- **drawqbpedge[parameter=value,...](startingNode,sa,endingNode,ea)**{**label**} =

  hrana mezi dvěma vrcholy pomocí kvadratické Bezierovy křivky, první a poslední bod jsou určeny vrcholy, prostřední bod určují dva úhly (ve stupních) vycházející z počátečního a koncového vrcholu
- **drawbpedge[parameter=value,...](startingNode,sa,sr,endingNode,ea,er)**{**label**}

  = hrana mezi dvěma vrcholy pomocí kubické Bezierovy křivky, počáteční a koncový bod jsou určeny vrcholy, druhý a třetí bod jsou definovány pomocí úhlů (ve stupních) a vzdálenostmi od krajních bodů
- **drawbcedge[parameter=value,...](startingNode,xs,ys,endingNode,xe,ye)**{**label**}

  = hrana mezi dvěma vrcholy pomocí kubické Bezierovy křivky, počáteční a koncový bod jsou určeny vrcholy, druhý a třetí bod určují souřadnice Kartézské soustavy
- **drawloop[parameter=value,...](Node)**{**label**} = hrana tvořící smyčku začínající i končící ve stejném vrcholu pomocí kubické Bezierovy křivky

- **drawline[parameter=value,...](x1,y1)...(xn,yn)** = lomená čára, která je definována dvěma a více body
- **drawcircle[parameter=value,...](x,y,d)** = kružnice se středem v souřadnicích *x,y* a průměrem *d*
- **drawrect[parameter=value,...](x0,y0,x1,y1)** = obdélník (čtverec) definovaný levým spodním a pravým horním rohem
- **drawoval[parameter=value,...](x,y,w,h,mr)** = ovál, který je definovaný souřadnicemi středu, šířkou, výškou a maximálním poloměrem zakulaceného rohu
- **drawpolygon[parameter=value,...](x1,y1)...(xn,yn)** = polygon, který je definovaný dvěma a více body, je možno mu zakulatit rohy
- **drawrpolygon[parameter=value,...](x,y)(n,r)** = pravidelný polygon, který je definovaný souřadnicemi středu, poloměrem a počtem vrcholů, je možno mu zakulatit rohy
- **drawccurve[parameter=value,...](x1,y1)...(xn,yn)** = vykreslí uzavřenou kubickou Bezierovu křivku, definovanou dvěma a více body, ve kterých křivka kontinuálně navazuje, řídící body jsou automaticky dopočítávány pro každý pár bodů
- **drawcurve[parameter=value,...](x1,y1)...(xn,yn)** = kubická Bezierova křivka, definována dvěma a více body, ve kterých křivka kontinuálně navazuje, řídící body jsou automaticky dopočítávány pro každý pár bodů
- **drawqbezier[parameter=value,...](x0,y0,x1,y1,x2,y2)** = vykreslí kvadratickou Bezierovu křivku, definovanou souřadnicemi tří kontrolních bodů
- **drawcbezier[parameter=value,...](x0,y0,x1,y1,x2,y2,x3,y3)** = vykreslí kubickou Bezierovu křivku, definovanou souřadnicemi čtyř kontrolních bodů

4 Analýza původního programu

Program *Editor a generátor grafů konečných automatů* Petra Hrubého byl vytvořen v roce 2008 v rámci bakalářské práce [3]. Zabývá se vytvářením grafů konečných automatů, které dovoluje vytvářet a upravovat pomocí přechodové tabulky nebo generovat náhodně vytvořené automaty. Program rovněž umožňuje zachovat a obnovovat program v souborech xml a dokáže vygenerovat jednoduchý soubor v LaTeXu.

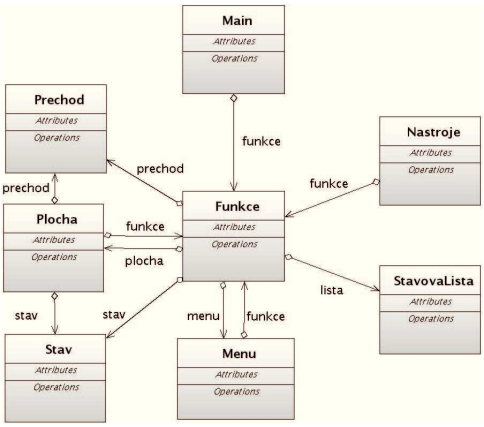
4.1 Implementace

4.1.1 Programovací jazyk Java

Program je naprogramován v programovacím jazyce Java. Jedná se o objektově orientovaný programovací jazyk, jenž vyvinula firma Sun Microsystems. Je to interpretovaný programovací jazyk, který generuje bytecode, který je spustitelný všude, kde je k dispozici interpret Javy, tzv. virtuální stroj Javy. Díky své jednoduché přenositelnosti na různé platformy se stal jedním z nejpoužívanějších programovacích jazyků. Používá se v čipových kartách, mobilních telefonech, desktopových počítačích i v rozsáhlých distribuovaných systémech.

4.1.2 Hlavní struktura programu

Program obsahuje 24 souborů s názvy tříd, které představují jednotlivé objekty a funkce programu. Hlavní třídou programu je třída *Funkce*. Ta zajišťuje běh celé aplikace, propojuje a zpracovává podněty z většiny ostatních tříd, zpracovává data a spouští jiné části programu (obr. 3). Jsou v ní obsaženy struktury (ArrayListy), které uchovávají v paměti programu *stavy* a *přechody* konečného automatu. Rovněž obsahuje paměť, do které se při každé změně uchovává celá struktura automatu a která umožňuje krokování vpřed a vzad při úpravách (až do 10 kroků zpět).



Obrázek 3: UML schéma základu programu

Třída *Stav* reprezentuje vrchol v grafu konečného automatu. Tyto stavy mají stálý tvar kružnice s možností změnit barvu čáry a textový popisek uvnitř kružnice. Obsahuje atributy souřadnic *x* a *y*, *poloměr kružnice*, *nazev*, jenž představuje popisek, *barva*, *typ* znamenající druh stavu, může nabývat 4 hodnot (normální, počáteční, koncový, počáteční i koncový) a *úhel*, který označuje směr šipky dovnitř u počátečního stavu.

Třída *Prechod* představuje hranu v grafu konečného automatu. Přechody v programu mohou být dvojího typu a to smyčka kolem jednoho stavu nebo čára (rovná, obloukovitá) mezi dvěma stavy. Přechod obsahuje informace o textovém popisku, straně a posunu popisku vůči čáře, barvě čáry a indexy stavů (uložených v ArrayListech), ve kterých přechod začíná a končí.

Program obsahuje i třídu *Text*, která reprezentuje textové popisky a pozici v souřadnicovém systému. Třídy Stav, Prechod i Text obsahují dále funkce pro ukládání do XML a generování LaTeX kódu.

4.2 Grafické rozhraní

Při spuštění programu se zobrazí okno, které je vytvořeno z JFrame. To se skládá ze tří respektive čtyř částí, jelikož prostřední část je rozdělena na dvě.

V horní části je umístěno *Menu*, které je vytvořeno pomocí JMenuBar, které obsahuje objekty typu JMenuSeparator (oddělovače) a JMenuItem (položky), které reagují na události kliknutí myší a spouštějí vybrané funkce viz. kapitola 4.4. Některé položky obsahují klávesové zkratky.

V prostřední části vlevo se nachází *Nástrojová lišta*, která je vytvořena z objektů JButton. Tyto tlačítka mají v sobě obrázek pro uživatelem snadné rozpoznání nástrojů, které umožňují *kreslit stavy*, *kreslit přechody*, *vybrat* na kreslící ploše objekty a *posouvat* jimi, *kopírování objektů*, *kreslení textů* a *mazání objektů*.

V prostřední části vpravo se nachází hlavní část vizuální části programu a to je *Kreslící plocha*. V ní se pomocí vybraných nástrojů z Nástrojové lišty dají kreslit stavy, přechody a texty, posouvat jimi či je otáčet a zakřivovat, kopírovat nebo smazat z kreslící plochy. Tyto úkony jsou závislé na událostech, které se vytvářejí pomocí levého tlačítka myši. Pravým tlačítkem myši můžeme vyvolat *Okna pro nastavení vlastností* stavů, přechodů a textů. Všechny tyto události zprostředkovává třída Funkce.

Ve spodní části nalezneme Stavovou lištu, která nám indikuje textovým popiskem, kterou nástrojovou funkci používáme a případně v jaké fázi se používaná funkce nachází např. „Vytvořit nový přechod(vyberte výchozí stav)“ nebo „Vytvořit nový přechod (vyberte koncový stav)“. U této části se vyskytla drobná chybička při vypisování textu, který se u některých popisů neobjevil celý, ale byl zkrácen např. „Vložit nový st...“.

4.3 Vykreslování objektů

Vykreslování automatu mají na starosti funkce ve třídě *Plocha*. U některých funkcí je potřeba dopočítat potřebné hodnoty, jako jsou průsečík přímky a kružnice nebo řídící bod *Kvadratické Bezierovy křivky*. Tyto výpočty zajišťuje třída *Matika*.

4.3.1 Stavy

Stavy jsou vykresleny jako kruh s bílým pozadím. Používají se k tomu funkce fillOval a drawOval. I když dovoluje třída Stav změnu poloměru kruhu, je v celém programu přednastavena pevná velikost. Uvnitř kruhu může být textový popisek, který je vycentrovaný vůči středu kruhu. Při označení stavu jako *koncový*, vykreslí se uvnitř kružnice obrys trošku menší kružnice. Když označíme stav jako *počáteční*, vykreslí se šipka o pevně nastavené délce směřující dovnitř stavu. Standardně se vykreslují barvy jako černá, v nastavení stavu si ale můžeme zvolit i žlutou, červenou, zelenou, modrou a šedou.

4.3.2 Přechody

Přechody mezi stavy se dají rozdělit na dva druhy podle toho, zda počáteční i koncový stav jsou shodné nebo se liší. První druh reprezentuje příkaz GasTeXu *drawloop* (smyčka) druhý *drawedge* (oblouk).

Smyčka je vykreslena jako Kubická Bezierova křivka je určena 4 body, 2 krajní body jsou shodné a 2 body se dopočítávají. I když můžeme uchopením a držením levého tlačítka myši smyčkou kolem stavu otáčet, její tvar vypadá vždy stejně a nejde jej nijak změnit. Jelikož má křivka neměnný tvar, kreslení textového popisu je rozděleno části, které reflektují posun popisku po délce křivky. Chybou, kterou jsem objevil, je změna strany popisku vůči smyčce, kde nejde nastavit možnost vpravo (dovnitř smyčky), i když ji nabídka upravení atributů křivky nabízí.

Druhým druhem přechodu je oblouk. Vede od jednoho stavu ke druhému a je vykreslován dvěma způsoby. Prvním způsobem je úsečka, když se zakřivení oblouku rovná nule. Druhým způsobem je Kvadratická Bezierova křivka. Té se dopočítá pomocí třídy Matika řídící bod, aby mohla být vykreslena. Rozdílem oproti GasTeXu je použití právě Kvadratické Bezierovy křivky, protože pro vykreslování křivky pomocí příkazu *drawedge* se používá Kubická Bezierova křivka.

Vykreslování přechodů probíhá jako první, protože začínají/končí ve středech stavů a stavy je pak překreslují. Popisek se na druhou stranu vykresluje až po vykreslení stavů, aby žádný text nebyl zakrytý. K oběma druhům přechodů se dokreslí šipka vyznačující jejich směr, její vrchol je tečnou s kružnicí stavu.

4.3.3 Texty

Posledním druhem objektu, který se kreslí na plochu je Text, definovaný souřadnicemi a textovým řetězcem, které určuje uživatel. Oproti popiskům stavů a přechodů jsou nakresleny tučným písmem. Texty rovněž nemohou být kopírovány ani přesunovány při zabrání více objektů. Mohou se přesunout pouze při samostatným posunu.

4.4 Funkce pro tvorbu konečných automatů

Program obsahuje funkce, které umožňují zadávat automaty, generovat náhodné nebo upravovat existující automaty pomocí tabulky přechodů. Po použití některé z těchto funkcí se smaže stávající automat a vygeneruje se automat nový, který se vykreslí v

řadách maximálně po třech stavech. Stavy jsou samozřejmě propojeny příslušnými přechody, které jsou popsány znaky zadané abecedy.

4.4.1 Zadávání automatu

Zadávání automatu umožňuje zadat deterministický či nedeterministický automat v tabulce přechodů. Funkce se spustí z hlavního menu *Generátor*. Objeví se okno, kde je třeba zadat počet stavů, vstupní abecedu a determinismus. Po potvrzení se objeví další okno, kde je potřeba zadat počáteční a koncové stavy. Je zde zobrazena i tabulka přechodů, kterou je potřeba vyplnit a nadefinovat tak požadovaný automat. Hlavními třídami, které zabezpečují tuto funkci jsou třídy *Zadavani*, *ZadavaniDeter* a *ZadavaniNedeter*.

4.4.2 Generování náhodného automatu

Generování náhodného automatu je pro uživatele mnohem zjednodušená verze zadávání automatu, protože nemusí vyplňovat tabulku přechodů. Opět ji spustíme z hlavního menu *Generátor* a po objevení okna stačí pouze vyplnit počet stavů, vstupní abecedu a determinismus automatu. Program už pak sám vygeneruje nějaký automat o zadaném počtu stavů s náhodně vytvořenými počátečními a koncovými stavy a také s náhodně vytvořenými přechody popsanými znaky zadané abecedy. To vše zajišťuje třída *Generovani*.

4.4.3 Úprava automatu pomocí tabulky přechodů

V menu *Soubor* najdeme funkci pro upravování stávajících automatů v tabulce přechodů. Uživatel může upravit počáteční i koncové stavy, názvy stavů a samozřejmě nastavit přechodové funkce mezi jednotlivými stavy. Práci s tabulkou přechodů umožňuje třída *Tabulka*.

4.5 Vstupní a výstupní funkce

Program umožňuje nadefinovat cestu k pracovnímu adresáři, který se ukládá do konfiguračního souboru. Při spuštění programu se pak načte cesta k požadovanému adresáři a při práci se soubory ji pak bude nabízet jako možnou alternativu pro jejich ukládání či načítání.

4.5.1 Soubory XML

Jelikož program neumí načítat GasTeX kód, probíhá zálohování (a znovu otevření) rozkreslených automatů do souborů XML. Hlavní kořenový element je *automat*, další elementy představující seznamy použitých objektů jsou *stavy*, *prechody* a *texty*. Tyto elementy mají pak jednotlivé elementy *stav*, *prechod* a *text*, které představují jednotlivé objekty s nastavenými parametry.

4.5.2 Kód v LaTeXu

Program umožňuje interpretovat automat i do LaTeXovém kódu. Dojde k přepočtu souřadnicového systému a výpočtu velikosti obrázku. Kód je ohraničen makrem picture, které definuje velikost obrázku. Jednotlivé objekty jsou v něm zobrazovány v pořadí: nejdříve stavy, pak přechody a nakonec samostatné texty.

4.5.3 Uložení do LaTeXu

Program dovoluje uložit automat i do souboru *.tex, pro uložení GasTeXových příkazů v LaTeXu. Při uložení dojde k podobným operacím jako ve výše zmíněné interpretaci pro LaTeX (kapitola 4.5.2) s tím rozdílem, že je zde doplněná hlavička dokumentu o načtení knihoven a globálního nastavení parametrů pro GasTeX viz. výpis 1.

```
\documentclass{article}
\usepackage[usenames]{color}
\usepackage{gastex}
\gasset{linewidth=0.21,AHdist=2.1,AHLength=2.25,AHlength=2.1}
\gasset{ATdist=2.1,ATLength=2.25,ATlength=2.1}
\gasset{Nw=8,Nh=8,Nmr=10,loopdiam=9}
```

Výpis 1: Definování hlavičky pro LaTeX v původním programu

4.5.4 Tabulka přechodů pro LaTeX

Uživateli umožňuje program vygenerování tabulky přechodových funkcí do kódu použitelného v LaTeXu z nakresleného automatu. V zásadě se nejedná o nic jiného než jen interpretaci přechodové tabulky, tak jak tomu bylo v kapitole 4.4.3. Rozdíl je, že tabulka není zobrazena graficky v Javě, ale je vygenerován její textový kód, který může být použit v LaTeXu.

4.6 Zhodnocení analýzy

Z analýzy původního programu v této kapitole vyplývá, že program je z programovacího hlediska naprogramovaný dobře, není proto nutné měnit hlavní strukturu tříd a spolupráci mezi nimi. Naopak je výhodné využít již naprogramovaného programu v programovacím jazyce Java, který stačí doplnit o další třídy a metody.

5 Překladač GasTeX kódu

Hlavním úkolem bakalářské práce bylo vytvořit překladač pro interpretaci GasTeXových příkazů.

Překladač [4] je program, který čte zdrojový program a překládá jej do cílového programu. Zdrojový program je napsán ve zdrojovém jazyce a cílový program v cílovém jazyce. Dnešní jazyky vyšší úrovně dokáží zjednodušit práci programátora a vyhnout se problémům, které vznikaly například při překladu do asemblerů, které jsou mnohem více závislé na strojovém kódu.

Hlavní výhodou moderních jazyků jsou tedy strojová nezávislost, která umožňuje přenositelnost na principiálně jiné architektury počítačů. Nabízejí ale i jiné možnosti, které pomáhají s redukcí logických chyb a laděním programu. Nevýhodou je pak velikost překladače a přeloženého kódu, stejně jako nižší rychlost překladu. Tyto nevýhody jsou ale většinou zanedbatelné s porovnáním výhod. Důležitými součástmi překladu jsou diagnostické zprávy, které informují uživatele o průběhu překladu, např. o výskytu nějaké chyby.

Překladače se používají při různých příležitostech. Mezi příklady můžeme uvést editory při formátování textů, kontrole správného syntaktického zápisu nebo doplňování či našeptávání klíčových slov. Mohou se taky používat v programech pro sazbu textů jako je TEX nebo je používáme běžně při prohlížení internetového obsahu v internetových prohlížečích.

5.1 Implementace překladače v JavaCC

Pro implementaci překladače byl zvolen program JavaCC. JavaCC je zkratka pro Java Compiler Compiler, což v překladu znamená Java Kompilátor Kompilátor. Je to program, který generuje lexikální a syntaktický analyzátor. JavaCC načte vstupní kód, který popisuje jazyk požadovaný k překladu, a vygeneruje výstupní kód v jazyce Java, jenž bude umět číst a analyzovat námi požadovaný překládaný jazyk. JavaCC je vhodný v případech, kdy má vstupní jazyk složitou strukturu a může být náročné vytvořit modul pro vstupní analýzu překladu. Vygenerované javovské soubory se pak dají přeložit normálním překladačem pro Javu. Přeložit jde jakýkoliv jazyk, který je popsán gramatickými pravidly. V našem případě překládáme všechna makra definovaná GasTeXem a několik maker LATEXu.

5.2 Sestavování gramatických pravidel

Pro fázi překladu, která se nazývá Lexikální analýza je potřeba vytvořit pravidla, díky kterým je možné načítat sekvence znaků nad zadanou abecedou.

Při lexikální analýze čteme znaky postupně zleva doprava ze zdrojového programu a sestavujeme posloupnosti lexikálních symbolů (tokens), které vytvářejí logickou posloupnost jako jsou klíčová slova jazyka, operátory, identifikátory nebo konstanty. Posloupnosti znaků, která odpovídá typu lexikálního symbolu se říká lexém.

```
\node(nazev)(10,20){popis}
```

Výpis 2: Ukázka příkazu *node* pro vykreslení vrcholu v GasTeXu

V ukázce příkazu GasTeXu (výpis 2) by se následující příkaz pro vytvoření vrcholu mohl načíst při lexikální analýze těmito lexikálními symboly:

1. *\node* - klíčové slovo
2. *(* - symbol levé kulaté závory
3. *nazev* - textový řetězec
4. *)* - symbol pravé kulaté závorky
5. *(* - symbol levé kulaté závorky
6. *10* - číselná hodnota
7. *,* - symbol čárky
8. *20* - číselná hodnota
9. *)* - symbol levé kulaté závorky
10. { - symbol levé složené závorky
11. *popis* - textový řetězec
12. } - symbol pravé složené závorky

Lexikální analýza zjistí pouze zda lexikální symbol může nebo nemůže být součástí zdrojového jazyka. Pokud by byly symboly zpřeházené, lexikální analyzátor by chybu nerozeznal. Správné pořadí se určuje až v následných analýzách.

Pro popis jazyka v JavaCC se používají regulární výrazy. Pokud by více regulárních výrazů mohlo popsat lexikální symbol, použije se pro analýzu ten, který popíše delší lexikální symbol. Pokud by popsané lexikální symboly byly pro oba regulární výrazy stejně dlouhé, pak se použije ten, který je nadefinován dříve.

```
TOKEN :
{
    <PARAMETR : "Nw"|"Nh"|"Nmr"|"Nframe"|"fillgray"|"Nfill"|"ExtNL"|"NLangle"|"NLdist"|"Nmarks"
    |"iangle"|"ilength"|"fangle"|"flength"|"rdist"|"Nmarks"|"Nadjustdist"|"Nadjust"|"sxo"|"syo"
    |"exo"|"eyo"|"curvedepth"|"loopwidth"|"loopheight"|"loopdiam"|"loopangle"|"loopCW"|"AHnb"
    |"AHdist"|"AHangle"|"AHLength"|"AHlength"|"ATnb"|"ATdist"|"ATangle"|"ATLength"|"ATlength"
    |"ELside"|"ELpos"|"ELdist"|"ELdistC"|"linegray"|"linewidth"|"dash"|"arcradius"|"polyangle"
    |"linecolor"|"fillcolor" >

   |< komentar: "%"(~["\n"])* >
}
```

```
TOKEN :
{
    <NUMBER: (("-")?(["0" - "9"])+ (["."] (["0" - "9"])+)*) >
  |<STRING: (["a"-"z","A"-"Z","0"-"9","_"])+ >
}
```

Výpis 3: Ukázka definovaných tokenů

V ukázce (výpis 3) vidíme některé nadefinované regulární výrazy, Jsou pro tokeny, které určují jedinečné názvy *všech parametrů příkazů GasTeXu*, pro *komentáře*, které jsou z levé strany označeny znakem procenta a pak se načítají všechny znaky dokud lexikální analyzátor nedojde do konce řádku. V ukázce vidíme také regulární výrazy rozpoznávající *čísla a názvy identifikátorů*.

5.3 Zpracování tokenů

Posloupnosti tokenů, které byly vytvořeny při lexikální analýze bylo potřeba nyní zpracovat tzv. *syntaktickou analýzou*. Pravidla pro syntaktický překlad jsou v JavaCC psány v EBNF, což je *Rozšířená verze Backusovy-Naurovy formy*, která se využívá k vyjádření bezkontextové gramatiky spolu s operátory regulárních výrazů.

Bezkontextová gramatika se používá pro překlad jazyků a nejčastěji se s ní setkáváme v syntaktické analýze zdrojových kódů programovacích jazyků. Přepisovací pravidla bezkontextové gramatiky jsou mnohem propracovanější než tomu je u regulárních výrazů, kdy nemůžeme například popsat několikanásobně vnořený text do závorek, protože nemůžeme zaručit stejný počet levých a pravých závorek.

Tokeny prochází syntaktickou analýzou a vytvářejí se hierarchické struktury, které dávají určitý význam, jako jsou například matematické výrazy, deklarace proměnných nebo i celý program. Tyto struktury (gramatické fráze) se pak vyjadřují často v derivačních nebo lexikálních stromech.

Pro analýzu se používají tzv. LL a LR gramatiky. LL gramatika se používá při analýze shora dolů. První písmeno L znamená, že se čte vstup postupně zleva doprava, druhé písmeno L pak znamená, že se derivace gramatiky také přepisují zleva doprava. U LR gramatik, které se používají při překladu zdola nahoru se pravidla derivují zprava doleva.

JavaCC používá analýzu shora dolů. Jednoduché LL(1) gramatiky vždy začínají na pravé straně terminálním symbolem. Jednička znamená, že se při analýze rozhoduje vždy podle jednoho symbolu. Tyto počáteční terminální symboly musí být různé. Obecná LL(1) gramatika nemá omezení, ale musí pro ni existovat rozkladová tabulka, která zaznamenává rozklad vstupní věty, nebo-li čísla rozkladových pravidel. Aby se jednalo o LL(1) gramatiku, nesmí u žádného záznamu být vícenásobně definovaná položka. V mnoha případech ale není gramatika, pro kterou chceme vytvořit syntaktický analyzátor typu LL(1).

JavaCC proto umožňuje používat nejen gramatiky typu LL(1), ale také vytvářet gramatiky, které jsou typu *LL(k)*. LL(k) gramatiky ovšem výrazně zpomalují zpracování vstupního souboru, a tak nám JavaCC dovoluje nadefinovat část gramatiky jako LL(k)

a zbytek gramatiky jako LL(1). LL(k) se definuje příznakem LOOKAHEAD, který je umístěn před lexikálními symboly gramatiky. Tato vlastnost JavaCC nám umožňuje vytvoření jednoduchého zápisu gramatiky bez výrazného zpomalení překladače.

V následujícím výpisu 4 je definována funkce pro syntaktickou analýzu zápisu GasTeXových parametrů ve tvaru „parameter=value,parameter=value,...“. Ty se načítají postupně, první parametr být definovaný musí, další parametry už nejsou povinné. Jednotlivá přiřazení hodnot parametrům jsou oddělena čárkami. Hodnoty mohou být trojího druhu a to *číselné hodnoty <NUMBER>*, *textové řetězce <STRING>* a *speciální hodnota Dash()*, která určuje spojitost čáry. Jelikož přiřazení hodnot není zcela popsáno gramatikou LL(1), je použita specifikace LOOKAHEAD(2), která dokáže nahlédnout dopředu a vybrat správnou variantu. Výstupem funkce je ArrayList, který nabývá hodnot třídy Parametr, která obsahuje název načteného parametru a jeho hodnotu.

```
ArrayList<Parametr> Parametr(): {Token tok; Token promenna=null; ArrayList<Parametr> list=
      new ArrayList<Parametr>(); Dash dash=null; int pom;} {
   tok=<PARAMETR> (<mezera>)* <rovnase> (<mezera>)* (LOOKAHEAD(2)(promenna=<
        NUMBER> {pom=0;})|LOOKAHEAD(2)(promenna=<STRING> {pom=1;})|LOOKAHEAD
        (2)(dash=Dash(){pom=2;}))
   {
      if (pom==0)list.add(new Parametr(tok.toString(),Double.parseDouble(promenna.toString())));
      if (pom==1)list.add(new Parametr(tok.toString(),promenna.toString()));
      if (pom==2)list.add(new Parametr(tok.toString(),dash));
    }
     (LOOKAHEAD(2)(<mezera>)* <carka> (<mezera>)* tok=<PARAMETR> (<mezera>)* <
          rovnase> (<mezera>)* (LOOKAHEAD(2)(promenna=<NUMBER> {pom=0;})|
          LOOKAHEAD(2)(promenna=<STRING> {pom=1;})|LOOKAHEAD(2)(dash=Dash(){pom
          =2;}))
      {
          if (pom==0)list.add(new Parametr(tok.toString(),Double.parseDouble(promenna.toString())
              ));
          if (pom==1)list.add(new Parametr(tok.toString(),promenna.toString()));
          if (pom==2)list.add(new Parametr(tok.toString(),dash));
       }
    )*
    {return list ;}
}
```

Výpis 4: Funkce pro definování parametrů

Mezi tokeny, které nejsou nijak dále zpracovávány patřily ve výpisu 4 lexikální symbol *rovnítka <rovnase>*, lexikální symbol *čárky <carka>* a také lexikální symboly pro *bílé znaky <mezera>*. V překladači JavaCC jde nastavit, aby se bílé znaky při načítání automaticky přeskakovaly, ale jelikož se bílé znaky musí načítat pro definování textových popisků příkazů GasTeXu, bylo nutné je zařadit do bezkontextové gramatiky a přeskakování nebylo umožněno.

Načítání parametrů se používá ve všech příkazech GasTeXu (příkazy popsány v kapitole 3.2). Pro každý příkaz je vytvořena bezkontextová gramatika a každý příkaz má vytvořenu svou vlastní třídu, kterou je interpretován.

Kromě příkazů GasTeXu umožňuje překladač načítat i univerzální neznámé příkazy, které nejsou definovány žádným klíčovým slovem. Tím umožňuje překladači načíst i

některé příkazy LaTeXu, které nejsou přímo podporovány, ale mohou se vyskytnout v překládaném textu.

Přeložit lze i LaTeXovský příkaz *put*, který vytvoří pravoúhlou plochu. Příkazy, které by tento příkaz mohl v LaTeXu interpretovat, jsou v našem editoru vykresleny pouze jako textové popisky.

Pro překladač byl rovněž nadefinován příkaz Komentář. Ten je uvozen z levé strany znakem procenta, pak mohou následovat jakékoliv znaky a z pravé strany jej ukončuje konec řádku.

Všechny tyto příkazy pak načítá jedna funkce *Program* (výpis 5), která je hlavní pro celý překladač. Jejím výstupem je ArrayList typu Object, který obsahuje všechny příkazy načtené překladačem. Tento seznam příkazů je pak dále zpracovávaný Editorem konečných automatů.

```
ArrayList<Object> Program(): {ArrayList<Object> prikazy=new ArrayList<Object>();Object
     prikazek= new Object();} {
    (
       ( prikazek=Gasset()
       | prikazek=RpNode() | prikazek=Node() | prikazek=Imark() | prikazek=Fmark() | prikazek=
           Rmark() | prikazek=NodeLabel()
       | prikazek=DrawEdge() | prikazek=DrawQBEdge() | prikazek=DrawQBPEdge() | prikazek=
           DrawBPEdge() | prikazek=DrawBCEdge() | prikazek=DrawLoop()
       | prikazek=DrawLine() | prikazek=DrawCircle() | prikazek=DrawRect() | prikazek=DrawOval()
            | prikazek=DrawPolygon() | prikazek=DrawRPolygon() | prikazek=DrawCCurve() |
           prikazek=DrawCurve() | prikazek=DrawQBezier() | prikazek=DrawCBezier()
       | prikazek=Prikaz() | prikazek=Put() | prikazek=Komentar()
       ){prikazy.add(prikazek);}
       | <mezera>
   )* <EOF>
    {return prikazy;}
}
```

Výpis 5: Hlavní funkce překladače

5.4 Generované soubory

Jak už bylo zmíněno, JavaCC generuje *.java soubory, které pak jsou schopny kompilovat a zpracovávat námi požadovaný zdrojový jazyk. Standardně vytváří tyto soubory:

Funkční soubory

1. SimpleCharStream.java - třída reprezentující tok vstupních znaků
2. Token.java - třída představuje vstupní token
3. TokenMgrError.java - třída vytváří chybové hlášení ze správce tokenů při lexikální analýze
4. ParseException.java - třída pro výjimku, která vytváří chybová hlášení při syntaktické analýze

Vlastní soubory (XXX - uživatelem zvolený vlastní název překladače)

1. XXX.java - hlavní třída představující překladač
2. XXXTokenManager.java - třída pro správu tokenů
3. XXXConstants.java - rozhraní, které sdružuje tokeny se symbolickými názvy

Funkční soubory se generují nezávisle na překládané gramatice. Generují se automaticky, když překladač JavaCC zjistí, že nejsou v cílovém adresáři. Překladač je nijak více nekontroluje, proto je celkem bezpečné, pokud bychom je chtěli po překladu do Javy modifikovat.

Oproti tomu *Vlastní soubory* jsou pokaždé kompilovány ze souboru *.jj. Úpravy po vygenerování jsou nedoporučované, jelikož při dalším vygenerování souborů by byly nové informace ztraceny a nahrazeny a musely by se opětovně specifikovat.

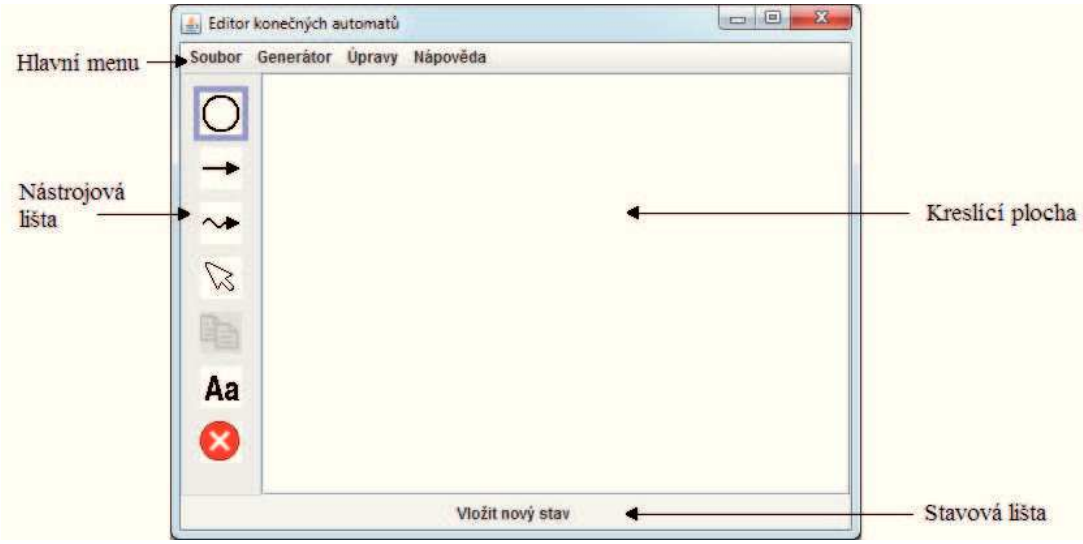
V nastavení překladače JavaCC, byla nastavena možnost JAVA_UNICODE_ESCAPE. Toto nastavení umožnilo místo generovaného souboru *SimpleCharStream.java* vygenerovat soubor *JavaCharStream.java*, který si umí poradit i se znaky znakové sady Unicode. Nastavující volby generátoru se definují v hlavičce *.jj souboru (výpis 6). Možnost IGNORE_CASE nastavená na false přikazuje překladači rozlišovat velká a malá písmena. Nastavením možnosti STATIC na false bylo umožněno opětovné volání konstruktoru překladače.

```
options {
    IGNORE_CASE = false;
    STATIC = false;
    JAVA_UNICODE_ESCAPE = true;
}
```

Výpis 6: Nastavování překladače JavaCC

6 Úprava původního programu

Nově upravený program se od původního programu na první pohled téměř vůbec neliší. Vizuálně v něm přibyla pouze jedna ikonka v nástrojové liště pro kresbu Kubické Bezierovy křivky o 4 řídících bodech (viz. obrázek 4). Hlavní menu má pak navíc na výběr položky „Načti GasTeX“ a „Parametry“. Ve skutečnosti byly ale provedeny změny ve všech třídách původního programu a přidány třídy nové, které zajišťují nejen chod překladače, ale také práci s příkazy a parametry GasTeXu a objektů, které tyto příkazy převádějí do grafické podoby. Z původního počtu 24 souborů reprezentující třídy programu se program rozrostl na 71.

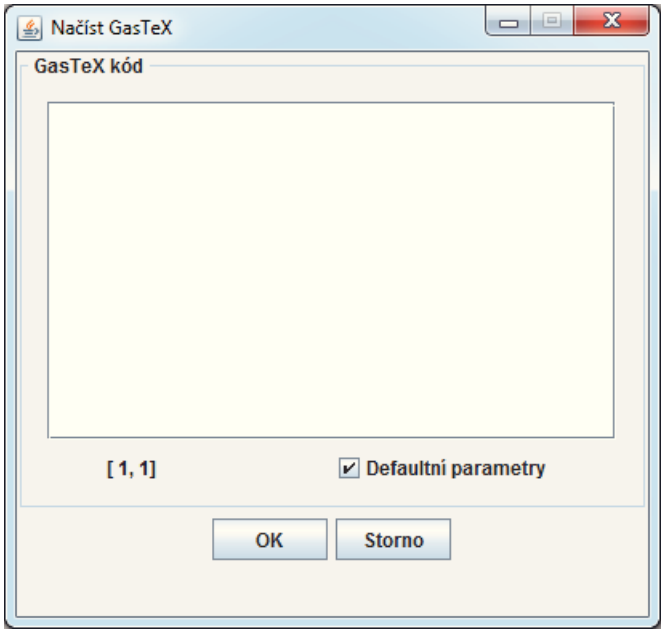


Obrázek 4: Hlavní okno programu

6.1 Propojení překladače s původním programem

Vygenerované soubory (viz. kapitola 5.4) byly přiloženy k ostatním zdrojovým souborům programu. Menu programu bylo rozšířeno o položku *Načti GasTeX*, která při kliknutí myší otevře okno pro načtení (obrázek 5).

Okno pro načtení GasTeXu zpracovává třída *NactiGasTeX*. Vizuálně se okno skládá z *JTextArea*, do kterého vkládáme v textové formě podporované příkazy LaTeXu respektive GasTeXu. V dolní části nám *JLabel* indikuje pozici kursoru uvnitř JTextArea. *JCheckBox* nám umožňuje zachovat globálně přednastavené parametry GasTeXu nebo nastavit *defaultní hodnoty*. Kliknutím na tlačítko *OK* spustíme překlad. Pokud během překladu nastane nějaká chyba, objeví se informační okno s výpisem chybové hlášky, podle které lze snadno najít příslušnou chybu a opravit ji. Chybovou hlášku program vytvoří i v případě,



Obrázek 5: Okno pro načtení příkazů GasTeXu

kdy se budeme pokoušet přeložit prázdný text. Při úspěšném přeložení se okno zavře a je vygenerován seznam všech načtených příkazů (ArrayList<Object>).

V ukázce zdrojového kódu (výpis 7) můžeme vidět zpracování textu, jehož znaky jsou nejdříve převedeny do proudu znaků (stream), který kódován znakovou sadou UTF-8. Proud znaků je pak zpracován překladačem. Pokud vše proběhne v pořádku, je překlad úspěšně dokončen a vytvořen seznam načtených příkazů, pokud nastala chyba, vytvoří výjimku, která je pak zobrazena a nastaví boolovskou proměnnou *prelozeno* na *false*, čímž zakáže zavřít okno pro načítání příkazů GasTeXu. Celkově indikujeme tři chyby při překladu, a to, jestli byl text správně přeložen do znakové sady nebo jestli nenastala chyba při lexikální anebo syntaktické analýze.

```
boolean prelozeno=true;
ArrayList<Object> seznam = new ArrayList<Object>();
try {
    prekladac preklad = new prekladac(new java.io.ByteArrayInputStream(text.getBytes("UTF-8")),
        "UTF-8");
    seznam=preklad.Program();
}
catch (ParseException e) {
    JOptionPane.showMessageDialog(new JFrame(), e.getMessage(), "Chyba!", JOptionPane.
        OK_OPTION);
    prelozeno=false;
}
catch (TokenMgrError e) {
```

```
        JOptionPane.showMessageDialog(new JFrame(), e.getMessage(), "Chyba!", JOptionPane.
            OK_OPTION);
        prelozeno=false;
    }
    catch (UnsupportedEncodingException e) {
        JOptionPane.showMessageDialog(new JFrame(), e.getMessage(), "Chyba!", JOptionPane.
            OK_OPTION);
        prelozeno=false;
    }
```

Výpis 7: Zpracování překládaného textu

6.2 Propojení paměťových struktur

V původním programu byly použity tři paměťové ArrayListy, které tvořily hlavní paměťovou strukturu programu. Sloužily pro ukládání stavů, přechodů a textů. Pro tyto seznamy byly vytvořeny funkce, které zajišťovaly jejich upravování a vykreslování. Výstupem překladače je seznam objektů reprezentující příkazy GasTeXu, jejichž struktura je velmi podobná těm skutečným v textové podobě (seznam parametrů a atributy).

Obě tyto paměťové struktury jsou v programu zachovány. Výhodou je, že se nemusely znovu vytvářet všechny funkce pro práci s hlavními objekty (stavy, přechody, texty). To ale přineslo i problémy s propojením obou struktur, aby data zůstala konzistentní. Pro jejich vzájemnou komunikaci byly vytvořeny funkce (hledání, upravování, vytváření, mazání), které přenášejí data ze seznamu příkazů GasTeXu do původních seznamů a naopak. Tím bylo zajištěno, že se data upravují v jedné i druhé paměťové struktuře zároveň. Většina těchto funkcí je definovaná ve třídě Funkce.

Původní paměťové struktury slouží pro práci v grafickém rozhraní a užívání funkcí automatu. Číselné hodnoty určující velikosti jsou v nich převedeny do počtu pixelů. Seznam všech příkazů je použit pro vstupní a výstupní operace.

6.3 Grafické zobrazení

Grafické zobrazení prodělalo mnoho změn. Původní reprezentace stavů a přechodů byla hodně statická a nedalo se upravovat moc parametrů. Stavy byly pouze kruhového tvaru a přechody taky měly své nedostatky, smyčce kolem jednoho stavu se nedala měnit velikost, obloukovitý přechod zase byl vykreslován Kvadratickou Bezierovou křivkou, kdežto v GasTeXu je reprezentován Bezierovou křivkou kubickou. Vylepšeny byly i pozice popisků křivek. Naopak odstraněna byla funkce pro minimální vzdálenost mezi stavy, jelikož načtený GasTeXový kód by stejně mohl tuto funkci obejít. Objekty textů zůstaly graficky nezměněny.

6.3.1 Stavy

Stavy jsou reprezentovány třídou *Stav*. V ní jsou uchovávány všechna data o daném stavu a funkce, které je zpracovávají. Ty jsou předávány do třídy *Plocha*, která je graficky zpracovává a vykresluje na Kreslící plochu programu. Třída Stav oproti původní verzi byla

téměř celá přepracovaná a přibyla v ní podpora pro nové parametry. Tím se musely změnit zároveň i vykreslovací funkce. Název a popisek již není sjednocen do jednoho parametru, ale byl rozdělen na dva samostatné atributy.

Třída Stav z GasTeXového hlediska reprezentuje příkaz *node* a graficky podporuje tyto parametry: *Nw, Nh, Nmr, Nmarks, iangle, ilenght, linecolor, fillcolor*. Popis parametrů najdete v kapitole 3.1. Všechny tyto parametry jdou upravit při kliknutí pravým tlačítkem na stav, kdy se objeví okno pro úpravu parametrů. Vložené parametry jsou zkontrolovány, zda odpovídají možným hodnotám a pokud ano, provedou se změny. V opačném případě se objeví chybová hláška.

Asi nejvíce vzhledově ovlivňující jsou parametry Nw, Nh a Nmr, které určují tvar stavu. Ten může být kruhovitý, obdélníkovitý nebo oválný (obdélník se zakulacenými rohy). To pak ovlivňuje spoustu funkcí, zejména na vykreslování přechodů (viz. kapitola 6.3.2).

Program podporuje graficky i příkazy *imark* (šipka dovnitř stavu) a *rmark* (vícečetné ohraničení stavu). Tyto příkazy jsou ale podporovány u každého stavu pouze jednou a je vykreslen v pořadí vždy ten poslední příkaz v seznamu. Ostatní výskyty těchto příkazů jsou uchovávány v seznamu načtených příkazů, ale graficky nejsou zobrazeny.

6.3.2 Přechody

Přechody jsou interpretovány třídou *Prechod*. Původně byly podporovány dva příkazy GasTeXu a to *drawedge* (rovná nebo obloukovitá křivka) a *drawloop* (smyčka kolem jednoho stavu). K tomu byl doprogramován příkaz GasTexu *drawbpedge* (volně nastavitelná křivka). Všechny tyto křivky jsou přeprogramovány na Kubickou Bezierovu křivku, což odpovídá i jejich interpretaci v GasTeXu. Oproti původnímu programu přibyla i funkce, která zajišťuje u smyčky, že ji změna velikosti nezakryje, ale dojde k jejímu posunutí.

Graficky jsou podporovány tyto parametry: *curvedepth, ELpos, ELside, loopwidth, loopheight, loopdiam, loopangle, ELdist, ELdistC, linecolor, sxo, syo, exo, eyo*. Popis parametrů najdete v kapitole 3.1. Tyto parametry jdou upravit obdobně jako parametry stavu (viz. kapitola 6.3.1). Parametry loopangle a curvedepth jsou měnitelné pouze uchopením křivky pomocí levého tlačítka myši a posunu objektem. Při kliknutí levým tlačítkem na volně nastavitelnou křivku ji nejde uchopit, ale zobrazí se okno pro nastavení zbývajících dvou řídících bodů Bezierovy křivky. Ty jsou nastavitelné od krajního bodu pomocí úhlu a vzdálenosti. Díky offsetovým parametrům (sxo, syo, exo, eyo) posunout počátek a konec křivky vůči středu stavu klidně i mimo něj.

Díky dynamickému měnění tvaru stavů, došly změnám i funkce na vykreslení šipek. Díky složitosti výpočtu průsečíku s křivkou byly změněny početní funkce na testovací, kdy se prochází křivka po pixelech a testuje se, jestli daný pixel náleží zároveň stavu. Pokud ano, pokračuje se na další pixel, pokud ne, je v daném bodě vykreslena šipka směrem do stavu. Podobným způsobem, kdy se prochází celá křivka, je testován i úchopový režim pro výběr křivky.

6.3.3 Popisky přechodů

Se změnou křivek bylo nutné měnit i popisky přechodů. Ty se nyní mnohem více přizpůsobují vykresleným křivkám a pomocí parametrů ELdist a ELdistC se dá nastavit i vzdálenost popisku od křivky.

6.4 Parametry

Se vznikem překladu a podporou interpretace příkazu *gasset* byla vytvořena třída Parametry, která nastavuje všechny parametry GasTeXu. Ty se dají buď načíst všechny pomocí překladače příkazem gasset nebo nastavit pouze podporované příkazy stavů (kapitola 6.3.1) a přechodů (kapitola 6.3.2) pomocí okna *Nastavení globálních parametrů*, které spustíme v menu *Generátor/Parametry*.

Tyto parametry se projevují v programu globálně, takže například, když si nastavíme parametry Nw, Nh a Nmr na námi požadované hodnoty, každý následující stav pak bude vytvořen s našimi hodnotami. To má určitě výhodu, když chceme vytvořit více shodných objektů se stejnými vlastnostmi, které tak nemusíme upravovat každému objektu samostatně.

Program nám dovoluje nastavit také defaultní hodnoty globálních parametrů (výpis 8).

```
private double Nw=8,Nh=8,Nmr=4;
private Nframe Nframe=new Nframe("y");
private double fillgray=0; private Nfill Nfill=new Nfill("n");
private ExtNL ExtNL=new ExtNL("n");
private double NLangle=90,NLdist=0;
private double iangle=180,ilength=5;
private double fangle=0,flength=5;
private double rdist=0.7;
private Nmarks Nmarks=new Nmarks("n");
private double Nadjustdist=1; private Nadjust Nadjust=new Nadjust("n");
private double sxo=0,syo=0,exo=0,eyo=0;
private double curvedepth=0;
private double loopwidth=8, loopheight=8, loopdiam=8,loopangle=90;
private loopCW loopCW=new loopCW("y");
private double AHnb=1,AHdist=1.41,AHangle=20,AHLength=1.5,AHlength=1.41;
private double ATnb=0,ATdist=1.41,ATangle=20,ATLength=1.5,ATlength=1.41;
private ELside ELside=new ELside("l"); private ELdistC ELdistC=new ELdistC("n");
private double ELpos=50,ELdist=1;
private double linegray=0;
private double linewidth=0.14; private Dash dash=new Dash();
private double arcradius=0,polyangle=0;
private Color linecolor= Color.black, fillcolor = Color.white;
```

Výpis 8: Defaultní nastavení globálních parametrů

6.5 Funkce s automaty

6.5.1 Zadávání a generování automatů

Funkce pro práci s automaty, které nám dovolují zadávání a generování deterministických a nedeterministických automatů, byly upraveny jen minimálně a funkčně zůstaly stejné jako v původním programu. Byly do nich pouze přidány a upraveny funkce, které pracují s třídami Stav a Prechod, protože je bylo nutné optimalizovat vůči nově nastaveným parametrům a propojit je s možností přidávat stavy a přechody do seznamu všech příkazů.

Do těchto funkcí byla rovněž vložena nabídka pro nastavení globálních parametrů, které můžeme ponechat na současných hodnotách anebo nastavit na defaultní.

6.5.2 Tabulky přechodů

V tabulce přechodů byly upraveny funkce tak, aby při načítání automatu do tabulky nebraly v potaz čárku mezi dvěma znaky abecedy. Vytvářel se tak nový sloupec tabulky což bylo nežádoucí. Byla taky opravena chyba, která měnila název stavu automatu při použití mezery v načtené abecedě.

Stejně jako v předchozí kapitole 6.5.1 bylo nutné přizpůsobit funkce pracující s třídami Stav a Prechod.

Po potvrzení tlačítkem OK je vytvořen a vykreslen nový automat, seznam všech načtených příkazů GasTeXu je rovněž zcela přegenerován.

6.6 Vstupní a výstupní data

Kromě funkce Kód tabulky (kapitola 6.6.1) se při práci se vstupními a výstupními daty pracuje se seznamem všech načtených příkazů GasTeX (ArrayList¡Object¿). V něm jsou ukládána data v podobě velmi blízké zápisu příkazů v textovém tvaru. Pořadí příkazů je totožné s postupným vytvářením automatu.

6.6.1 Kód tabulky

Program umožňuje vygenerovat kód tabulky do zápisu v LaTeXu. Tato funkce je velmi obdobná jako *Tabulka přechodů* (kapitola 6.5.2) s tím rozdílem, že v Tabulce přechodů dochází k interpretaci tabulky do grafického uživatelského rozhraní, kde je možnost tabulku měnit a přegenerovat stávající automat. Kód tabulky pouze vytvoří okno, kde vypíše tabulku přechodů v textové podobě.

Stejně jako v Tabulce přechodů byly objeveny a opraveny stejné problémy, které vytvářela čárka a mezera v načítané abecedě z popisků přechodů.

6.6.2 Kód automatu

Možnost *Kód automatu* vytváří okno s textovým polem, v němž jsou v textové podobě vygenerovány příkazy GasTeXu (LaTeXu). Jsou v něm interpretovány všechny příkazy, které

byly vytvořeny při práci s programem nebo při načítání příkazů z překladače. Mohou se v něm objevit všechny příkazy GasTeXu včetně těch, které nejsou programem graficky podporovány, jelikož seznam načtených parametrů je uchovává všechny. Uživatel má možnost si příkazy zkopírovat do vlastního LaTeXového dokumentu.

6.6.3 Export

Funkce export je obdobná jako funkce Kód automatu. Vytváří příkazy GasTeXu ze seznamu všech načtených příkazů. Funkce ale vytváří navíc i jednoduchou strukturu LaTeXového dokumentu, jsou do něj vloženy makra vytvářející hlavičku dokumentu užívání grafických balíků GasTeXu. Pro makro definující obrázek se dopočítává velikost vytvořeného automatu.

6.6.4 XML a DTD

Pomocí značkovacího jazyku XML se uchovávají a obnovují námi vytvořené konečné automaty.

Zkratka XML znamená Extensible Markup Language, který byl vyvinut a standardizován konsorciem W3C. Je to značkovací jazyk, který nemá vlastní značky (tagy), ty si ale můžeme sami vytvořit. Značky se definují v souboru DTD (Document Type Definition), kterým se dá automaticky kontrolovat struktura používaného XML dokumentu.

Při ukládání XML souboru se vytvoří nový dokument, ve kterém se vytvoří hlavička dokumentu definující znakovou sadu UTF-8. Hned za ní se vytvoří kořenový element s názvem *automat*. Do něj se pak vkládají další elementy, které představují jednotlivé příkazy GasTeXu. Do nich jsou vkládány další elementy s hodnotami atribudů a parametrů.

Při načítání XML dokumentu se použije soubor DTD (výpis 9), podle kterého se soubor zkontroluje a načtou se jednotlivé značky do stromové struktury, která se dále zpracovává. Tato metoda se nazývá DOM.

Pomocí funkcí *getElementsByTagName*, *getChildNodes*, *getAttributes* a *getNamedItem* se získávají jednotlivá data, která jsou pak interpretována v Editoru konečných automatů.

```
<?xml version='1.0' encoding='UTF−8'?>
<!−−− Kořenový element −−>
<!ELEMENT automat (drawbcedge|drawbpedge|drawcbezier|drawccurve|drawcircle|drawcurve|
    drawedge| drawline|drawloop|drawoval|drawpolygon|drawqbedge|drawqbezier|drawqbpedge|
    drawrect|drawrpolygon|fmark|gasset|imark|komentar|neznamyprikaz|node|nodelabel|put|rmark
    |rpnode)∗>
<!ELEMENT drawbcedge (hodnoty, parametry?)>
<!ELEMENT drawbpedge (hodnoty, parametry?)>
...
<!ELEMENT hodnoty EMPTY>
<!ATTLIST hodnoty
    nodename CDATA #IMPLIED
    x NMTOKEN #IMPLIED
    y NMTOKEN #IMPLIED
```

Výpis 9: Část souboru DTD

7 Závěr

Přínosem programu je snadné načtení a upravování GasTeXových příkazů v grafickém uživatelském prostředí a jeho následný export zpět do GasTeXu. Tím je program užitečný hlavně při úpravách složitějších grafů konečných automatů, kdy si již nedokážeme představit v textovém kódu, jak bude program vykreslen. Spolu s původními funkcemi pro tvorbu a úpravy konečných automatů se z programu stává mnohem komplexnější program.

V dalších verzích by se mohl editor doplnit o další algoritmy, které pracují s automaty. Mohl by se například rozšířit o funkce provádějící minimalizace automatu nebo kontrolu zda zadané slovo by bylo konečným automatem přijato či nikoliv. Také by se mohlo do programu doplnit několik grafických funkcí, které by rozšířily podporované makra GasTeXu. V tomto směru bych byl ale opatrný, jelikož další gafické funkce a ovládání by mohly uživateli v mnohém ztížit ovládání programu.

Pavel Byma

8 Reference

[1] Doc. RNDr. Petr Hliněný, Ph.D., *Úvod do Teoretické Informatiky*, 2006,
http://www.cs.vsb.cz/kot/download/uti2006/UTI-text06.pdf [citováno 19. dubna 2011].

[2] GasTeX manuály a knihovny,
http://www.lsv.ens-cachan.fr/~gastin/gastex/ [citováno 19. dubna 2011].

[3] Petr Hrubý, *Editor a generátor grafů konečných automatů*, Ostrava, 2008. 27 s. Bakalářská práce na Fakultě elektrotechniky a informatiky VŠB-TU Ostrava na katedře informatiky.

[4] Dr. Ing. Miroslav Beneš, *Překladače, http://www.cs.vsb.cz/behalek/vyuka/pjp/skripta/skr-mb.pdf* [citováno 19.dubna 2011].

[5] The JavaCC FAQ,
http://www.engr.mun.ca/~theo/JavaCC-FAQ [citováno 1. května 2011]

[6] DTD Tutorial,
http://www.w3schools.com/dtd/default.asp [citováno 1. května 2011]

[7] JasTeX Manuel d'utilisation,
http://www.lsv.ens-cachan.fr/~gastin/JasTeX/docs/francais/book1.htm
[citováno 19. dubna 2011].

A Příloha

Veškeré přílohy jsou uloženy na CD, které je přiloženo k bakalářské práci.

A.1 Program

/Program/editor.jar

A.2 Uživatelský manuál

/Uzivatelsky manual/index.html

A.3 Programátorská dokumentace

/Programatorska dokumentace/index.html

A.4 Vlastní text bakalářské práce

/Text bakalarske prace/

Obsahuje pdf soubor s touto bakalářskou prací a také LaTeXový dokument, ze kterého bylo toto pdf generováno.