Masarykova univerzita
Fakulta informatiky

Transformace bezkontextových gramatik

Bakalářská práce

Zuzana Krejčová

Brno, jaro 2011

Prohlášení

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

Vedoucí práce: doc. RNDr. Jiří Barnat, Ph.D.

Poděkování

Chtěla bych poděkovat vedoucímu práce, doc. RNDr. Jiřímu Barnatovi, Ph.D., za trpělivost a dobré rady a svému příteli, Vratislavu Podzimkovi, za nekonečnou trpělivost, podporu a užitečné poznámky k této práci. Dále textu *Tao of programming* a jeho autorům za připomenutí, že nemám očekávat dokonalost.

Shrnutí

Tato práce se zabývá tvorbou desktopové aplikace pro práci s bezkontextovými gramatikami. Textová část práce představuje hlavní body teorie formálních jazyků a blíže se zabývá bezkontextovými gramatikami, zejména jejich transformacemi. Podává přehled o technologiích a nástrojích použitých při vývoji této aplikace a věnuje se problémům, které při práci bylo nutné vyřešit, a jejich řešením. Popisuje možnosti a funkce aplikace, uživatelské rozhraní i další možnosti vývoje.

Klíčová slova

Bezkontextová gramatika, CFG, Java, XML, Bison, LaTeX

Obsah

Úvod

Teorie formálních jazyků je jednou z nejpodstatnějších částí informatiky jako vědního oboru. Několikrát denně využíváme výsledků zkoumání v této oblasti, většinou aniž bychom si to jakkoli uvědomovali. Programy, se kterými pracujeme, jsou vybudované pomocí programovacích jazyků — a ty nejsou nic jiného než formální jazyky popsané formálními gramatikami. Než si zobrazíme internetovou stránku v prohlížeči, musí ji dostat syntaktický analyzátor. Vyhledávání výrazů v textu je záležitost jednoduchého automatu.

Přestože se bez formálních jazyků, automatů a gramatik dnes neobejdeme, většina populace nepotřebuje žádné zvláštní znalosti v tomto oboru. Valná většina těch, kteří nějaké takové vzdělání potřebují, jsou studenti různých škol informatického zaměření. Například studenti Fakulty informatiky Masarykovy univerzity (dále FI MU), kteří musí povinně zvládnout jeden z předmětů *IB005 Formální jazyky a automaty I* a *IB102 Automaty a gramatiky*. Právě těm je primárně určena grafická desktopová aplikace, která je v rámci této bakalářské práce vyvíjena. Tato aplikace umožňuje uživatelům pracovat s bezkontextovými gramatikami — vytvářet nové a generovat náhodné gramatiky, nebo je načíst ze souboru, dále je transformovat a ukládat v několika různých formátech pro další potřeby. Je tedy vhodnou učební pomůckou pro nemalou skupinu studentů, která má právě s transformacemi bezkontextových gramatik potíže. Další využití program snadno najde v Laboratoři zpracování přirozeného jazyka na FI MU, případně u odvážlivců experimentujících s vývojem vlastního programovacího jazyka.

Text této práce je rozdělen podle témat do několika kapitol. První představuje čtenáři základy teorie formálních jazyků, důvody jejího vzniku a několik důležitých osobností zabývajících se touto problematikou, přiblíží Chomského hierarchii jazyků, resp. gramatik. Následuje kapitola věnovaná bezkontextovým gramatikám, jejich ty-

pům a transformacím a praktickému využití — k tomu patří i podkapitola věnovaná programu Bison, který pro bezkontextovou gramatiku dokáže vytvořit syntaktický analyzátor. Třetí kapitola se zabývá použitými technologiemi. Obhajuje výběr programovacího jazyka použitého pro vývoj aplikace a vysvětluje využití XML, HTML a LaTeXu. Samotná aplikace je pak popsána v následující, čtvrté kapitole. Podrobněji popisuje doposud implementované funkce programu a jeho možné využití. V této části jsou představeny původní návrhy a plány, k nimž je připojen současný stav a výsledné řešení aplikace. Součástí je podkapitola rozebírající další možný vývoj aplikace a to nejen uživatelského rozhraní, ale i přidaných funkcí, případně doprovodných miniaplikací.

Kapitola 1

Teorie formálních jazyků

Přestože je teorie formálních jazyků jednou z nejznámějších a nejdůležitějších oblastí teoretické informatiky, svůj původ má v lingvistice a filosofii jazyka a (jako snad u všeho v informatice) základy tvoří matematika, respektive algebra. Lingvisty pochopitelně zajímá, jak a čím jsou přirozené jazyky tvořeny, co mají společného, čím se liší. Člověku je blízké ve všem hledat nebo uměle tvořit systém, a tak není divu, že i syntaxe jazyka byla podrobena zkoumání. Syntax v lingvistice zahrnuje vztahy mezi slovy, pravidla pro správné tvoření vět a větných celků. Představuje právě takový zkoumatelný systém. Pro formální jazyky je syntax souborem pravidel, podle kterých lze ze základních stavebních kamenů jazyka vytvořit korektní větu daného jazyka. Na rozdíl od formálních jazyků, u kterých k analýze věty stačí syntax, jazyky přirozené jsou často mnohoznačné a k určení významu věty potřebujeme navíc i sémantiku výrazů a slovních spojení.

K vývoji této teorie významně přispělo vícero osobností různého zaměření [2]. Svůj podíl na něm mají matematik *Axel Thue* a *Emil Post*, kteří přišli s pojmem formálního systému nebo gramatiky, nebo *Alan Turing*, který formalizoval algoritmy pomocí svého Univerzálního (Turingova) stroje. Podstatnou roli hrálo i modelování různých fenoménů v rámci vývojové biologie — například L-systémy *A. Lindenmayera*, s jejichž pomocí je možné modelovat růst rostlin.

Jednou z hlavních osobností, které se problematikou jazykového systému a jeho reprezentace zabývaly, je *Noam Chomsky* [6], na poli informatiky známý převážně pro svou hierarchii jazyků a gramatik. Chomsky se v oblasti teorie formálních jazyků zajímal hlavně o vyjadřovací schopnost jednotlivých typů formálních jazyků a jejich schopnost reprezentace přirozeného jazyka. Poznatky v této oblasti

jsou obzvlášť důležité pro stále se vyvíjející a zdokonalující aplikace pro zpracování přirozeného jazyka — nástroje pro strojové čtení a interpretaci textu, programy určené pro Turingův test umělé inteligence apod.

1.1 Formální jazyk, gramatika, abeceda

Formální jazyk je konečná nebo nekonečná množina slov nad nějakou konečnou abecedou, kde abeceda je libovolná konečná množina symbolů a symbol nemusí být pouze jeden znak (písmeno, číslice, interpunkční znaménko…). Symbolem může být libovolně dlouhý (konečný) řetězec znaků. Prázdné slovo se obvykle značí malým řeckým písmenem ε nebo malým e, někdy je možné se setkat i s označením λ. Slovo jazyka se pak skládá z těchto symbolů a je také konečné[1]. Někdy se můžeme setkat i s označením věta na místě slova, což o něco lépe evokuje syntax přirozených jazyků.

Běžně se používají tyto způsoby zápisu formálního jazyka:

- výpisem všech slov jazyka,
- množinovým zápisem — např. $\{a^n b^n | a, b \in \Sigma^*, n \geq 0\}$,
- automatem, Turingovým strojem,
- gramatikou.

Není možné říct, že je jeden způsob všeobecně nejlepší — každý má své výhody a nevýhody a pro každý existuje situace, kdy je vhodné takový přístup použít. Bylo by poněkud naivní a marné pokoušet se formální jazyk zapsat výčtem všech jeho slov, pokud je nekonečný. Naopak pro jazyk, který je konečný a obsahuje jen několik slov, které nemají nic moc společného, by bylo neefektivní konstruovat automat nebo gramatiku. Pokud chceme o jazyce něco dokazovat, může se nám pro změnu hodit množinový zápis, jak jej známe z algebry.

Pro tuto práci budou nadále podstatné hlavně gramatiky, automaty jsou uvedeny spíše pro úplnost — zvědavý čtenář si může doplnit vzdělání z některého ze zdrojů, uvedených v sekci Literatura.

1. Existují teorie pracující s nekonečným slovem, pro tuto práci nejsou však podstatné.

Automat lze stručně popsat jako pomyslný stroj (nebo matematický model), který po přečtení vstupního řetězce symbolů rozhodne, zda dané slovo, resp. vstupní řetězec, patří do jazyka, pro který byl sestrojen. Existuje několik typů automatů s odlišnými vlastnostmi i definicemi, v přehledu jsou zmíněny v podkapitole Chomského hierarchie.

Máme-li nadále pracovat s gramatikami, bylo by vhodné definovat, co je gramatika v kontextu teorie formálních jazyků. Nehledě na její typ, gramatika popisuje pomocí souboru pravidel, jak je možné tvořit slova, která patří do konkrétního jazyka. Obvykle říkáme, že gramatika generuje jazyk. Formálně definujeme gramatiku takto:

„Gramatika G je čtveřice (N, Σ, P, S), kde

- N je neprázdná konečná množina neterminálních symbolů (stručněji: neterminálů).
- Σ je konečná množina terminálních symbolů (terminálů) taková, že $\mathrm{N} \cap \Sigma = \emptyset$.
  Sjednocením N a Σ obdržíme množinu všech symbolů gramatiky, kterou obvykle označujeme symbolem V.
- $P \subseteq V^*NV^* \times V^*$ je konečná množina pravidel. Pravidlo (α, β) obvykle zapisujeme ve tvaru $\alpha \rightarrow \beta$ (a čteme jako „α přepiš na β“).
- $S \in N$ je speciální počáteční neterminál (nazývaný také kořen gramatiky).“ [8]

Gramatika tedy musí mít minimálně jeden neterminál vzhledem k tomu, že musí nutně mít kořen. Dalo by se samozřejmě předpokládat, že gramatika bez kořene by jednoduše generovala prázdný jazyk, budeme se však držet výše uvedené tradiční definice. Gramatika může mít prázdnou abecedu, tedy množinu terminálů, nicméně to by vedlo buď k prázdnému jazyku nebo k jazyku s jediným slovem, ε, takže se s tím v praktickém využití příliš nesetkáme. Levá strana pravidla gramatiky je tvaru V^*NV^*, je to tedy řetězec symbolů gramatiky, z nichž minimálně jeden musí být neterminál. Na pravou stranu pravidla nejsou taková omezení kladena — jde jednoduše o libovolně dlouhý řetězec symbolů, třeba i délky 0. Pak se jedná o

tzv. ε-pravidlo, speciální pravidlo vyjímající řetězec na levé straně z větné formy[2]. V závislosti na omezeních kladených na jednotlivá pravidla rozlišujeme několik základních typů gramatik, uspořádaných v Chomského hierarchii gramatik.

1.2 Chomského hierarchie gramatik

Hned na úvod je třeba poznamenat, že Chomského hierarchie gramatik definuje pouze čtyři základní typy gramatik — frázové, kontextové, bezkontextové a regulární. Na později vyčleněné typy (indexované, stromové, deterministické bezkontextové, gramatiky pro rekurzivní jazyky apod.) nereflektuje. Není však cílem této práce zdokumentovat veškeré myslitelné typy gramatik, tento nedostatek tedy můžeme přehlížet.

Noam Chomsky v polovině 20. století rozdělil formální gramatiky (a jimi generované jazyky) na čtyři typy podle jejich vyjadřovací schopnosti. Definice všech těchto gramatik mají shodný základ, liší se pouze v požadavcích kladených na pravidla. Platí, že čím výraznější je omezení pravidel gramatiky, tím menší vyjadřovací sílu má třída jazyků odpovídající tomuto typu gramatiky. Dále platí, že gramatiku vyššího typu je zároveň možné klasifikovat i jakýmkoli typem nižším. Tedy například gramatika typu 2 je zároveň i typu 1 a 0, nemusí už ale být typu vyššího, tedy 3.

- *Typ 0 — frázové gramatiky*
  Je možné se setkat i s označením gramatiky bez omezení, právě podle absence jakékoli restrikce pravidel. Generují jazyky akceptované Turingovým strojem, někdy zvané rekurzivně spočetné.

- *Typ 1 — kontextové gramatiky*
  Pravidla tohoto typu gramatiky jsou omezená délkou řetězců na pravé a levé straně pravidla. Gramatika nesmí obsahovat tzv. zkracující pravidlo, musí tedy platit, že pravá strana pravidla je minimálně délky levé strany pravidla. Výjimku tvoří pravidlo pro přepsání kořene na ε, pokud se kořen nenachází

2. Větná forma je označení pro libovolný řetězec symbolů, který lze z kořene gramatiky odvodit.

na pravé straně žádného z pravidel. Jazyky těmito gramatikami generované se nazývají kontextové, akceptuje je lineárně ohraničený Turingův stroj.

- *Typ 2 — bezkontextové gramatiky*
  Bezkontextové gramatiky nadále zužují možnosti pravidel — na levé straně pravidla je povolen pouze jediný neterminál. Ve většině definic se navíc uvádí omezení na ε-pravidla, obdobně jako u kontextových gramatik, nicméně případných ε-pravidel se dá snadno zbavit transformací na ekvivalentní[3] bezkontextovou gramatiku bez ε-pravidel, která toto omezení splňuje. Tyto gramatiky generují jazyky akceptované zásobníkovým automatem, jazyky bezkontextové.

- *Typ 3 — regulární gramatiky*
  Regulární gramatiku je možné definovat dvojím způsobem — jako pravolineární nebo levolineární. V obou případech platí požadavek jediného neterminálu na levé straně pravidla. U pravolineárních je pravá strana definovaná jako jeden terminál, po kterém může následovat jeden neterminál, s obvyklou výjimkou ε-pravidla. U levolineární gramatiky může naopak neterminál terminál předcházet. Omezení ε-pravidel se dá obejít stejně jako u bezkontextových gramatik (vzhledem k výše zmíněné inkluzi typů gramatik). Jazykům takto generovaným se říká regulární a akceptuje je konečný automat.

Praktické využití mají dnes hlavně gramatiky regulární a bezkontextové, z velké části díky nepříliš složité syntaktické analýze. Chomsky původně předpokládal, že je možné přirozený jazyk modelovat právě pomocí bezkontextových gramatik, to bylo ovšem později vyvráceno.

3. Gramatiky jsou ekvivalentní, pokud generují stejný jazyk.

Kapitola 2

Bezkontextové gramatiky

Gramatiky všeobecně i některé významné typy byly představeny v předchozí kapitole, kde byly uvedeny také jejich definice. Na tomto místě tedy už pojem bezkontextové gramatiky nebudu znovu zavádět. Tato kapitola se věnuje vlastnostem těchto gramatik, jejich transformacím a aplikacím.

2.1 Vlastnosti

U bezkontextových gramatik můžeme zkoumat několik důležitých vlastností, které často ovlivňují, jak snadno se s nimi dá pracovat. Jsou to hlavně tyto:

- Gramatika je *levorekurzivní*, pokud má levorekurzivní neterminál. Tedy pokud existuje $A \in N$ takové, že $A =>^+ A\beta$ [1], kde $\beta \in \Sigma^*$. Obdobně lze definovat pravou rekurzi.
- Gramatiku *cyklickou* můžeme považovat za speciální případ gramatiky levorekurzivní. Pokud platí, že existuje $A \in N$ takové, že $A =>^+ A$, pak je gramatika cyklická. Naopak zaručeně necyklická je v případě, že neobsahuje žádná ε-pravidla (vyjma dříve zmiňované výjimky pro kořen) a jednoduchá pravidla (pravidla typu $A \to B$, kde $A, B \in N$).
- Vlastnost *sebevložení* definujeme obdobně jako rekurzi. Gramatika má tuto vlastnost, pokud existují $A \in N$ a $u, v \in \Sigma^*$ takové, že $A =>^+ uAv$. Pokud má každá gramatika generující jazyk L vlastnost sebevložení, má ji i jazyk L — to nastane právě tehdy, když L není regulární.

1. Kde $=>^+$ je relace netriviálního odvození, viz [8] (str. 5).

- Gramatika je *vlastní*, právě když neobsahuje žádné nepoužitelné symboly, ε-pravidla ani cykly. Vzhledem k definici cyklické gramatiky výše můžeme také říct, že gramatika je vlastní, pokud neobsahuje nepoužitelné symboly, ε-pravidla a jednoduchá pravidla.

- *Jednoznačnost* gramatiky je vlastnost obzvlášť důležitá při syntaktické analýze a jejím praktickém využití. Pokud je gramatika nejednoznačná, existuje více způsobů, jak z kořene gramatiky odvodit konkrétní slovo — to samozřejmě výrazně komplikuje analýzu a většina nástrojů tento problém řeší výběrem prvního možného pravidla. Jazyk označíme za nejednoznačný, resp. mnohoznačný, pokud neexistuje jednoznačná gramatika, která by jej generovala. Pro jednoznačné jazyky (mají alespoň jednu jednoznačnou gramatiku) je možné místo nejednoznačné gramatiky najít jednoznačnou — ty jsou ovšem často velice nepřehledné a několikanásobně složitější.

Bezkontextové gramatiky se mohou, krom všeobecné formy, vyskytovat také v několika normálních formách. Známé jsou Chomského a Greibachové normální formy a zařadíme sem i redukovanou gramatiku. Každou bezkontextovou gramatiku je možné převést do těchto normálních forem, s výjimkou gramatik generujících prázdný jazyk.

- *Redukovaná* gramatika je taková, která neobsahuje dva konkrétní typy symbolů — nenormované, které není možné jakýmkoli počtem kroků přepsat na řetězec terminálů, a nedosažitelné, ke kterým se odvozováním z kořene ani není možné dostat.

- *Chomského normální forma*, dále CNF, povoluje pouze pravidla dvou tvarů: $A \to BC$ a $A \to a$, kde $A, B, C \in N$) a $a \in \Sigma$). Povolenou výjimkou je pravidlo $S \to \varepsilon$, pokud je S kořen a nenachází se na pravé straně žádného pravidla.

- *Greibachové normální forma*, dále GNF, má pravidla tvaru $A \to a\alpha$, kde $A \in N$), $\alpha \in N^*$) a $a \in \Sigma$), opět s výjimkou ε-pravidla pro kořen.

Dále se můžeme setkat s tzv. LL, resp. LL(k) gramatikami. Tohoto speciálního typu bezkontextových gramatik se využívá při deterministické syntaktické analýze (shora dolů). Název značí levou derivaci a analýzu zleva doprava, k reprezentuje počet symbolů, které je třeba dopředu načíst[2]. Snad nejvyužívanější jsou gramatiky typu LL(1).

Existují algoritmy pro převod bezkontextové gramatiky na LL(1) gramatiku, protože jsou však LL gramatiky vždy jednoznačné a ne z každé gramatiky lze nejednoznačnost odstranit, přirozeně stejně tak není možné převést všechny gramatiky. Tyto gramatiky jsou také zásadně nelevorekurzivní.

S LL gramatikami úzce souvisí funkce FIRST a FOLLOW. Mámeli pravidlo $N \to \alpha$, pak FIRST(α) počítá množinu terminálů, kterými může α začínat (po přepsání patřičných neterminálů podle dostupných pravidel). Pokud $\alpha = \varepsilon$, FIRST(α) = $\{\varepsilon\}$. Funkce FOLLOW(N) počítá množinu terminálů, které mohou po tomto neterminálu následovat (včetně ε).

Pro LL gramatiky dále platí, že neobsahují žádný konflikt FIRST-FIRST nebo FIRST-FOLLOW. Tedy množiny těmito funkcemi počítané jsou disjunktní. První typ konfliktu nastává, mají-li dvě množiny počítané funkcí FIRST pro dva různé neterminály některé prvky shodné. Druhý konflikt nastává, pokud se pro jedno pravidlo prolínají množiny FIRST a FOLLOW. Při převodu gramatik na LL(1) je třeba na to dát pozor a konflikty odstranit.

2.2 Transformace

Pomocí různých transformací je možné u bezkontextových gramatik dosáhnout několika zajímavých a užitečných vlastností, případně se zbavit vlastností nežádoucích a gramatiku zpřehlednit. Gramatika původní je vždy jazykově ekvivalentní s transformovanou gramatikou — generují stejný jazyk. Ve své práci využívám následujících transformací:

- eliminace nenormovaných neterminálů,
- eliminace nedosažitelných symbolů,

2. Obdobou jsou LR(k) gramatiky — pravá derivace, analýza zleva doprava.

- eliminace ε-pravidel,
- eliminace jednoduchých pravidel,
- převedení do CNF,
- odstranění levé rekurze,
- převedení do GNF,
- levá faktorizace,
- rohová substituce.

Využívám verze algoritmů tak, jak je uvádí učební texty *Formální jazyky a automaty I* [8] — formální definici je možné nalézt právě tam, zde uvádním pouze krátký slovní popis principu algoritmů — a *Regulární a bezkontextové jazyky II* [9].

Eliminace nenormovaných neterminálů

V tomto algoritmu je postupně plněna množina všech těch neterminálů, které lze přepsat na řetězec terminálů. Po naplnění je množina prohlášena za novou sadu neterminálů a z pravidel se odstraní ta, která využívají jiné než tyto neterminály. Algoritmus lze chápat i jako test na (ne)prázdnost jazyka — zůstane-li množina prázdná, nelze žádný neterminál, tedy ani kořen, přepsat na řetězec terminálůa jazyk touto gramatikou generovaný je prázdný.

Eliminace nedosažitelných symbolů

Obdobně jako u předchozího algoritmu — plníme množinu symbolů těmi, které lze z kořene dosáhnout, neterminály i terminály. Ty, které po skončení tohoto cyklu v množině nenajdeme, jednoduše vyjmeme z neterminálů a terminálů a odebereme z pravidel ta, která je využívají.

Eliminace ε-pravidel

Nejprve je nutné zjistit, které neterminály je možné přepsat na ε. Poté pro každé pravidlo s takovými neterminály na pravé straně přidáváme i všechny variace pravidla s vypuštěním některých nebo všech takových neterminálů — krom případu, kdy by tím vzniklo nové ε-pravidlo. Pokud je nutné upravovat i pravidla pro kořen, zavádí se nový kořen S' s jedinými dvěma pravidly $S' \rightarrow S|\varepsilon$.

Eliminace jednoduchých pravidel

I eliminace jednoduchých pravidel pracuje s obdobným principem jako již zmíněné transformace. Pro každý neterminál X tvoříme množinu N_X těch neterminálů, na které se může jednoduchým pravidlem přepsat některý neterminál z této množiny, přičemž na počátku do množiny vkládáme právě X. Poté opět upravujeme pravidla — všechna nejednoduchá pravidla neterminálů ze zkonstruovaných množin přidáme k těm neterminálům, které jej mají ve své množině. Tedy pro všechna nejednoduchá $N \rightarrow \alpha$ přidáme pravidlo $X \rightarrow \alpha$ všem X takovým, že $N \in N_X$. Tímto nahradíme pravou stranu jednoduchého pravidla za nejednoduchou pravou stranu cílového neterminálu.

Převedení do CNF

Algoritmus předpokládá, že vstupní gramatika je nejen bezkontextová, ale též vlastní. Postup je velice prostý: všechna pravidla, která už jsou ve tvaru vyhovujícím Chomského normální formě, necháme tak, jak jsou. Ostatní pravidla jsou jednoho ze dvou typů (gramatika je vlastní, nemáme tedy jednoduchá pravidla) — $N \rightarrow X_1X_2$, kde $X_1 \in N$ a $X_2 \in \Sigma$ nebo naopak. V takovém případě terminál na pravé straně nahradíme novým neterminálem <X_i>, tento neterminál přidáme do množiny neterminálů a vytvoříme nové pravidlo <X_i> $\rightarrow$ X_i. Druhá možnost je, že pravidlo je tvaru $N \rightarrow \alpha$, kde $\alpha \in V^*, |\alpha| \geq 3$. Pak první symbol necháme, resp. nahradíme novým neterminálem jako v předchozí možnosti, a z celého zbytku řetězce vytvoříme další neterminál: z $N \rightarrow X_1X_2...X_n$ se stane $N \rightarrow$ <X_1><$X_2...X_n$> a

vytváříme nová pravidla <$X_2 ... X_n$> → <X_2><$X_3 ... X_n$> pro všechny nově vznikající neterminály.

Odstranění levé rekurze

Algoritmus v podstatě nahrazuje levou rekurzi za pravou a odstraňuje levorekurzivní smyčky — situace, kde $A \rightarrow B\alpha$ a zároveň i $B \rightarrow A\beta$. V prvním kroku uspořádáme množinu neterminálů nějakým vhodným způsobem — například $\{A_i, \ldots, A_n\}$, kde $A_i \rightarrow A_j\alpha$. Pak procházíme tyto neterminály a jejich pravidla v určeném pořadí a každý neterminál N, který nalezneme na prvním místě na pravé straně a který už jsme takto prošli, nahradíme všemi řetězci α, kde $N \rightarrow \alpha$ je pravidlo této gramatiky. Následně se zbavujeme přímé levé rekurze, tedy situace $N \rightarrow N\alpha$ právě nahrazením za pravou rekurzi.

Převedení do GNF

Pokud chceme gramatiku převést do GNF tímto algoritmem, musí být především vlastní a bez levé rekurze. Opět potřebujeme seřadit neterminály, tentokrát ale na způsobu velice záleží, tedy: $\{A_1, \ldots, A_n\}$, kde platí, je-li pravidlo $A_i \rightarrow A_j\alpha$, pak $i < j$. Poté iterativně, od předposledního prvku k prvnímu, procházíme toto uspořádání a každý již vyřešený neterminál N, vyskytne-li se na začátku pravé strany pravidla pro aktuální neterminál, nahradíme všemi řetězci α, kde $N \rightarrow \alpha$ je pravidlo gramatiky.

Posledním krokem je nahrazení terminálů, které se nenachází na začátku řetězce, novými neterminály stejným způsobem jako při převodu do CNF.

Levá faktorizace

Levá faktorizace, které se jinak říká také vytýkání díky podobnosti s touto matematickou operací, je algoritmus určený k řešení FIRST-FIRST konfliktu při transformaci na LL gramatiku. To je situace, kdy dvě pravidla pro jeden neterminál začínají stejným symbolem. Problém vyřeší nahrazení těchto konfliktních pravidel za jediné, které začíná onou opakující se částí řetězce, za tu je vložen nový neterminál. Tento nový neterminál pak umožníme přepsat na zbývající části

kolizních pravidel (tedy bez kolizního začátku).

Vstup: $G = (N,\ \Sigma,\ P,\ S)$
Výstup: $G = (N,\ \Sigma,\ P,\ S)$ bez FIRST-FIRST konfliktu
  for all $A \to \alpha\beta_1|\alpha\beta_2|\ldots|\alpha\beta_n$ **do**
    odstraň z P $A \to \alpha\beta_1|\alpha\beta_2|\ldots|\alpha\beta_n$
    přidej do N A'
    přidej do P $A' \to \beta_1|\beta_2|\ldots|\beta_n$
    přidej do P $A \to \alpha A'$
  end for

Rohová substituce

Tato procedura je ve své podstatě zjednodušené převedení gramatiky do GNF — opomíjíme odstranění levé rekurze a záměnu terminálů uvnitř pravidla za neterminály. Pouze substituujeme neterminál na začátku pravidla za řetězce, na které lze tento neterminál přepsat. Využívá se při transformaci na LL(1) gramatiku.

2.3 Aplikace

Pravděpodobně nejznámější praktická aplikace bezkontextových gramatik je jejich využití pro popis syntaxe programovacích jazyků. Na rozdíl od gramatik regulárních, pomocí bezkontextových už dokážeme generovat i slova typu $a^n b^n$. Dokážeme jimi tedy popsat například správně uzávorkované výrazy a podobné struktury. Pokud už takový jazyk máme definovaný, využijeme bezkontextové gramatiky s nástroji pro tvorbu syntaktických analyzátorů (parserů) a překladačů — takovými jsou mimo jiné Yacc (Yet Another Compiler Compiler) [11], jeho nástupce GNU Bison a jeho obdoba JavaCC. Jako speciální zápis bezkontextové gramatiky můžeme chápat i DTD (Documet Type Definition) [15].

GNU Bison

GNU Bison je nástroj pro generování různých LR syntaktických analyzátorů z bezkontextových gramatik, zapsaných ve formátu se základem v Backus-Naurově formě (BNF) [14]. Jazykem implementace

i výstupního analyzátoru je C, resp. C++, nicméně testuje se už i verze pro Javu. Ve stručnosti lze říct, že tomuto nástroji dáte bezkontextovou gramatiku jazyka, pro který potřebujete syntaktický analyzátor, a on vám vrátí soubor s implementací tohoto analyzátoru v C++. Většinou nestačí jen syntaktický analyzátor, chete-li opravdu něčeho prakticky dosáhnout. Je třeba vytvořit také lexikální analyzátor — k tomu slouží například nástroj Lex.

Formát vstupního souboru pro GNU Bison je velice specifický a přesný popis toho, co je kde dovoleno či přikázáno, je k nalezení na WWW stránkách programu, v jeho manuálu [16]. Zde uvádím jen základní strukturu.

```
%{
Prolog
%}
Deklarace programu Bison
%%
Pravidla gramatiky
%%
Epilog
```

V prologu a epilogu bývá umístěn kód v jazyce C (C++), deklarace programu obsahují názvy a datové typy symbolů, případně pořadí operátorů, a sekce pravidel gramatiky obsahuje pravidla gramatiky zapsané ve variaci na BNF. Zatímco v BNF by pravidlo vypadalo takto:

<neterminál> ::= pravidlo | "terminál1" | "terminál2" | "+"

Ve formátu pro Bison bychom napsali:

neterminál : pravidlo | TERMINÁL1 | "terminál2" | '+' ;

Obvykle se jednotlivá pravidla pro přehlednost zapisují pod sebe tak, aby svislá čára byla pod dvojtečkou — obdobně bývá umístěn středník.

Kapitola 3

Použité technologie

Vývoj jakéhokoli programu vyžaduje využití různých technologií, v této kapitole stručně představuji, čeho jsem využívala při práci na výsledné aplikaci, hlavní výhody a důvody pro tuto volbu.

Je tedy nevyhnutelné podat základní informace o programovacím jazyce Java a s tím spojeným prostředím NetBeans. Dále se věnuji XML formátu, s ním pracujícím nástrojům a dalším formátům od něj odvozeným.

3.1 Java

„Java je programovací jazyk a počítačová platforma, vydaná společností Sun Microsystems v roce 1995.“[1] [18] Jak tvrdí Oracle na svých stránkách, je všestraná, efektivní, přenosná a bezpečná. Najdeme ji mimo jiné i v tiskárnách, navigačních systémech, lékařských zařízeních — jinými slovy téměř všude.

S tímto hodnocením zajisté nemusí každý naprosto souhlasit. Přenositelnost Javy je ale nepopiratelná a považuji ji za nejvýraznější výhodu této technologie. Pro Javu také hovoří masivní vývojářská komunita — velké množství lidí si ji oblíbilo a podílí se na jejím neustálém vývoji. Navíc je dokumentace Java API snadno dostupná a dobře napsaná.

Mnozí Javě vyčítají zdlouhavý zápis i těch nejjednodušších programů a jistou pomalost startu i běhu. Nelze popřít, že například v jazyce Python lze stejný program napsat na pouhý zlomek řádků a že třeba program zapsaný v C nebo C++ bude rychlejší, i kdyby jen

1. „Java is a programming language and computing platform first released by Sun Microsystems in 1995.“

kvůli tomu, že není třeba spouštět žádný virtuální stroj. Na Windows je ovšem vývoj v jazyce Python poněkud ztížen a píšete-li v C nebo C++, musíte kód kompilovat pro každou platformu zvlášť. Jakmile se Java Virtual Machine jednou načte, ani s rychlostí programů v Javě to není tak špatné. Používá-li programátor inteligentní vývojové prostředí, jakým je například NetBeans, je navíc velice jednoduché vytvořit grafické uživatelské rozhraní a namapovat libovolné funkce k libovolným prvkům.

Výběr jazyka pro mě byl s ohledem na výše zmíněné velice jednoduchý. Mám-li na systému Windows, který stále ještě preferuji, vytvořit aplikaci s grafickým rozhraním, připadá mi Java jako nejschůdnější cesta. Předpokládám-li navíc použití programu uživateli různých systémů, je volba jasná.

NetBeans

Vývojové prostředí NetBeans je další z pozitiv při práci s Javou. Dokáže vám nabídnout snad všechno, co byste mohli během vývoje programu potřebovat. Nápověda k syntaxi Javy a doplňování kódu neuvěřitelně zrychluje práci, vestavěný editor grafického rozhraní usnadňuje programování desktopových aplikací nad knihovnou Java Swing. Pracujete-li na týmovém projektu, není problém používat Subversion či jiný verzovací systém přímo z NetBeans.

Toto prostředí dokáže velké množství drobností dělat za vás. S jistou nadsázkou by se dalo říct, že pro vývoj aplikace pomocí NetBeans stačí základní znalost syntaxe Javy a přibližná představa toho, co vlastně chcete.

3.2 XML

XML je zkratkou pro eXtensible Markup Language, dnes už poměrně dobře zavedený značkovací jazyk udržovaný sdružením W3C a vycházející z jednoduchosti jazyka HTML a komplexních popisných možností jeho základu, metajazyka SGML. Umožňuje snadný popis uchovávaných informací a jejich výměnu mezi aplikacemi. Toho dosahuje mimo jiné i díky striktnímu oddělení struktury dat od jejich vizuální prezentace v dokumentu. O vzhled prezentovaných infor-

mací se už starají jiné jazyky a nástroje. Korektnost dokumentu ve smyslu použití správných značek a atributů je možné zkontrolovat například oproti DTD nebo XML Schema.

Data ve formátu XML připomínají velice čistý HTML kód. Například jednoduchá knihovna by mohla vypadat nějak takto:

```
<?xml version="1.0" encoding="UTF–8"?>
<knihovna>
  <kniha isbn="nejaky kod">
    <nazev>Tao of programming</nazev>
    <autor>Geoffrey James</autor>
    <rok>neznamy</rok>
  </kniha>
  <kniha isbn="nejaky jiny kod">
    <nazev>XML v prikladech</nazev>
    <autor>Benoit Marchal</autor>
    <rok>2000</rok>
  </kniha>
</knihovna>
```

XML Schema

Pokud chcete mít jistotu, že k vám dorazí data v konkrétní podobě, je dobré definovat XML Schema dokumentu. Schema je soubor s definicemi jednotlivých značek a atributů. Omezuje a upřesňuje, kde je možné konkrétní značky a atributy použít, kolikrát se na daném místě mohou vyskytovat a jaká data v nich mohou být uložena. Například pro výše uvedenou knihovnu bychom mohli určit, že každá kniha musí mít atribut *isbn* a řetězec v něm bude tvaru *„3 cifry–0 až 5 cifer–až 7 cifer–až 6 cifer–právě 1 cifra“*, musí mít alespoň jednoho autora a právě jeden název a rok vydání, kterým je čtyřmístné číslo začínající 0, 1 nebo 2.

Zdrojový kód XML Schema bývá poměrně rozsáhlý a i jednoduché XML Schema pro zmíněnou knihovnu by zabralo minimálně celou stránku, proto zde příklad neuvádím. Schema pro XML popis gramatiky je součástí zdrojových souborů aplikace vyvíjené k této práci, je tedy možné se podívat tam.

XML Schema samotné je XML dokument.

Document Object Model

Ukládáte-li data ve formátu XML, musíte se také rozhodnout, jakým způsobem je z tohoto formátu budete načítat a jak je do něj budete ukládat. Možností je několik, pro tuto práci jsem si vybrala DOM — Document Object Model. DOM je rozhraní z dílny W3C, které umožňuje pracovat s XML dokumentem jako se stromovou strukturou. K jednotlivým značkám, atributům, textovému obsahu i ke komentářům přistupujeme jako k uzlům takového stromu. Veškeré takto zpřístupněné hodnoty je možné upravovat i mazat, případně další vytvářet.

Pokud ovšem chcete vytvořit nad daty nový XML soubor, je pravděpodobně jednodušší přímý zápis do souboru.

3.3 LaTeX a (X)HTML

LaTeX je balík maker vystavěný nad programem TeX D. E. Knutha, který použití sázecího systému TeX zjednodušuje. Využívá se pro sázení odborných prací, obzvlášť vhodný je pro dokumenty obsahující matematické vzorce nebo speciální abecedy. Příkazy pro LaTeX se zapisují přímo do textu, který je třeba vysázet.

HTML, HyperText Markup Language, je jazyk pro popis dokumentu hojně využívaný pro tvorbu WWW stránek. Vychází z metajazyka SGML a obdobně příkazům pro TeX umožňuje značení struktury obsahu i jeho vizuální podoby.

XHTML, Extensible HyperText Markup Language, je značkovací jazyk vyvíjený skupinou W3C. Předpokládá se, že nahradí starší jazyk HTML. XHTML byl vytvořen jako XML verze jazyka HTML — typ XHTML Transitional sice ještě dovoluje vizuální značky, jejich použití se ale výrazně nedoporučuje. Je s HTML zpětně kompatibilní, je už ale třeba dbát na správné uzavření (párových) značek, korektní zanoření a množství dalších specifik.

Kapitola 4

Transformátor gramatik

Výsledkem této bakalářské práce je především desktopová aplikace řešící vybrané transformace bezkontextových gramatik. Hlavním účelem této aplikace je pomoct studentům kurzů *IB005 Formální jazyky a automaty I* a *IB102 Automaty a gramatiky* osvojit si základní transformace bezkontextových gramatik[1]. Program umožňuje provést veškeré v těchto předmětech zmiňované transformace, student si tedy může ověřit své řešení libovolné z nich.

Jak již bylo zmíněno v úvodu k této práci, své místo by mohl tento program najít i v Laboratoři zpracování přirozeného jazyka na FI MU nebo u odvážlivců a zvídavých jedinců, kteří by se pokoušeli o vytvoření vlastního (ne nutně programovacího) jazyka, nebo jednoduše analyzátoru pro ověření nějakých dat — i pak se může hodit program pro transformaci gramatik.

Aplikace je vyvíjena pod licencí GNU GPL v3 [19], zdrojový kód je k dispozici k nahlédnutí na domovských stránkách projektu na serveru Google Code, konkrétně na adrese http://code.google.com/p/transformator-gramatik/.

Práce s programem je jednoduchá a přímočará. Uživatel může existující gramatiku načíst ze souboru nebo zadat novou přímo v programu, případně si může programem nechat vygenerovat náhodnou gramatiku. Je implementováno načítání z textového a XML souboru — více se načítání dat věnuji v sekci o reprezentaci gramatik. Dále může uživatel provést libovolné množství požadovaných transformací — výpis transformované gramatiky může obsahovat pouze tuto gramatiku nebo i všechny gramatiky, na které bylo nutné gramatiku původní transformovat (například při převodu na vlastní

1. Eliminace ε a jednoduchých pravidel, nepoužitelných symbolů, levé rekurze, převedení na CNF a GNF, levá faktorizace, rohová substituce a identita.

gramatiku, kdy je nutné odstranit ε a jednoduchá pravidla a nepoužitelné symboly), a pomocné množiny (jako například N_ε).

Po provedení požadovaných transformací je možné gramatiku uložit jako text (takto je možné uložit i netransformovanou gramatiku). Pokud tento formát nepostačuje, lze transformovanou gramatiku uložit (exportovat) jako XML soubor, vstupní soubor pro GNU Bison, (X)HTML stránku nebo zdrojový soubor pro LaTeX.

Samotné grafické uživatelské rozhraní této aplikace je poměrně přímočaré. Transformace gramatik jsou přístupné z menu Transformace, tamtéž je i zaškrtávací políčko pro volbu stylu výpisu. Uživatel si vybírá mezi krátkým výpisem, ve kterém je pouze transformovaná gramatika, a výchozí volbou delšího výpisu, kdy si může projít i veškeré jednoduché transformace, které bylo třeba provést, a všechny pomocné množiny. Rozdíl je vidět například při transformaci na vlastní gramatiku, kdy jednoduchý výpis obsahuje pouze výslednou gramatiku a v obsáhlejším výpisu je i gramatika po odstranění nepoužitelných symbolů a další po eliminaci ε a jednoduchých pravidel a také množiny N_ε a N_X pro každý neterminál X.

Z menu Gramatika jsou přístupné volby vytvoření či načtení nové gramatiky, uložení či export gramatiky do podporovaných formátů a ukončení běhu programu. Kromě exportu mají všechny tyto volby přidělenou klávesovou zkratku pro uživatelovo pohodlí. Menu O Programu nabízí dvě volby — zobrazení stručných informací o programu a zatím neimplementované, vzhledem k jednoduchosti programu však nepříliš potřebné, nápovědy.

4.1 Návrhy a řešení

Reprezentace gramatik

Důležitou otázkou hned na začátku vývoje bylo, jak reprezentovat gramatiku uvnitř programu a vně. Rozhodnout o vnitřní reprezentaci bylo snazší — gramatika je samostatná třída s neterminály, terminály, pravidly a kořenem uchovávanými v atributech třídy. Obdobně uchovávám informaci o některých provedených transformacích — eliminace ε-pravidel, jednoduchých pravidel a odstranění nepoužitelných symbolů. Zjednodušují se tím následující transformace.

```
public class Gramatika
{
    protected TreeSet<String> neterminaly;
    protected TreeSet<String> terminaly;
    protected String koren;
    protected TreeMap
    <String,HashSet<ArrayList<String>>> pravidla;
    protected boolean bezEpsilon = false;
    protected boolean bezNepouzitelnych = false;
    protected boolean bezJednoduchych = false;
    ...
}
```

Rozhodování o konkrétní reprezentaci gramatiky vně programu, resp. jaký typ vstupu bude program přijímat a jakým způsobem bude gramatiky ukládat do souboru, bylo o trochu složitější. Bylo třeba vzít v úvahu pohodlí uživatele. Hlavním způsobem zápisu gramatiky je následující, zapsaný v textovém souboru.

({A,B,C,S},{a,b,c},S,P)
P:
A –> ab | A | SC | ε
B –> abc | ABC | [eps]
S –> C | ABc

Toto má několik omezení a pravidel. Protože se jedná o čistý text, takto zvolený právě pro nejvyšší míru jednoduchosti zápisu, je nutné ho převést na objekt typu Gramatika „ručně“. Analýza by byla velice složitá pro situaci, kdy bychom měli symboly s_1, s_2 a $s_1 s_2$ a pravidlo, které by některý neterminál přepisovalo právě na $s_1 s_2$. Jedná se o po sobě následující první dva symboly, nebo o symbol třetí? Je tedy nutné se držet pravidla, že libovolný symbol je vždy právě jeden znak. Bylo by možné toto obejít, nicméně to by znamenalo o něco složitější zápis pro uživatele. Znak ε je možné, jak je vidět na příkladu, zapsat buď právě tímto znakem, v Unicode znak 0x03B5, nebo řetězcem [eps].

Zápis gramatiky musí mít předepsaný formát — v kulatých závorkách množina neterminálů, množina terminálů, kořen a P, značící pravidla. Následuje řádek *P:* a poté další řádky specifikující vlastní

pravidla. Ta musí obzvlášť dodržovat oddělení levé a pravé strany znaky –> a oddělení jednotlivých pravidel pomocí | . Pravá strana pravidla nesmí být prázdná.

Kromě textového souboru může být gramatika čtena i z XML soubou. Soubor gramatikaXmlSchema.xsd obsahuje XML Schema korektního XML souboru pro tento program. Následující příklad ukazuje velice jednoduchou gramatiku zapsanou jako XML soubor.

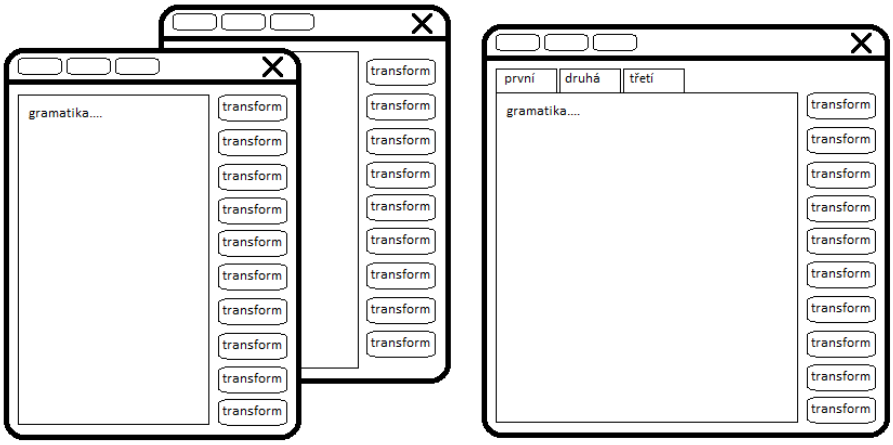
```
<?xml version="1.0" encoding="UTF-8"?>
<gramatika>
  <symboly>
    <symbol name="START" typ="n" />
    <symbol name="stop" typ="t" />
  </symboly>
  <pravidla>
    <pravidlo>
      <leva_strana name-ref="START" />
      <prave_strany>
        <prava_strana>
        </prava_strana>
        <prava_strana>
          <symbol name-ref="START" />
          <symbol name-ref="stop" />
        </prava_strana>
      </prave_strany>
    </pravidlo>
  </pravidla>
  <koren name-ref="START" />
</gramatika>
```

Pokud zvolíme tento způsob zápisu gramatiky, můžeme pro symboly použít i delší řetězce, což je obrovská výhoda. Z XML souboru je také snadné data převést do jiného formátu pomocí XSLT — pokud aplikace uživateli nenabízí vhodný formát pro uložení gramatiky, může být vhodná volba právě XML a následná XSL transformace. Nevýhodou, pro většinu uživatelů pravděpodobně skoro až nepřekonatelnou, je zdlouhavý ruční zápis. Použitím vhodného editoru lze zápis zjednodušit, pořád se to ale nevyrovná jednoduchosti textového formátu gramatiky.

Znak ε, lépe řečeno ε-pravidlo, se v tomto formátu značí prázdnou značkou <prava_strana /> — velice příhodné, uvážíme-li, že takové pravidlo opravdu je jakousi prázdnou pravou stranou pravidla, pravidlem, které neterminál vyjme z větné formy.

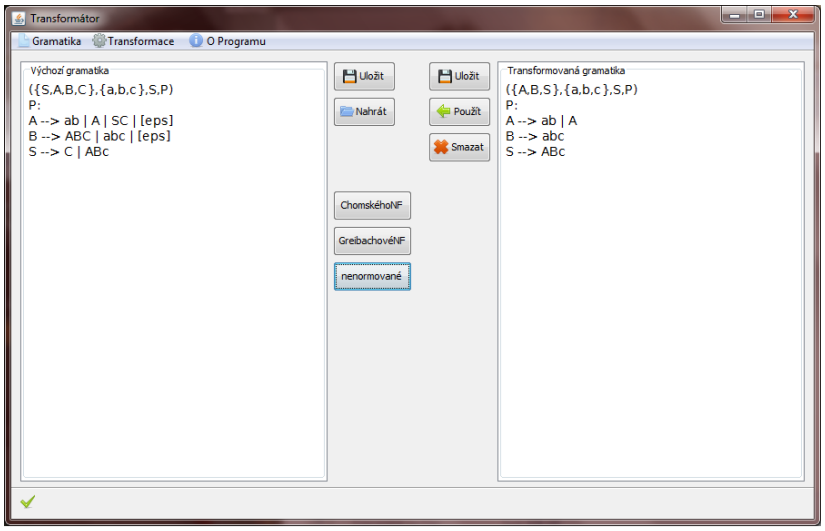
Vzhled aplikace, grafické rozhraní

Nebylo snadné se rozhodnout, jak by měla aplikace vypadat — struktura jejího rozhraní i styl. Druhý problém nakonec řeší Java sama, lépe řečeno třída UIManager z balíku Swings. Uživatel systému Windows uvidí okno programu tak, jak je zvyklý z Windows, uživatel linuxových distribucí uvidí typický styl grafického rozhraní pro tuto platformu.



Obrázek 4.1: Návrhy s více okny a panely

Jak ale uspořádat zobrazované informace? Návrhů bylo několik. Poněkud nešťastné jednoduché zobrazení gramatiky v okně, kde by tato gramatika byla po transformaci přepsána novou, nebo o něco vhodnější využití několika panelů v jednom okně (přístupných přes záložky) a řešení se zobrazením nového okna pro každou novou gramatiku — viz obrázek 4.1.

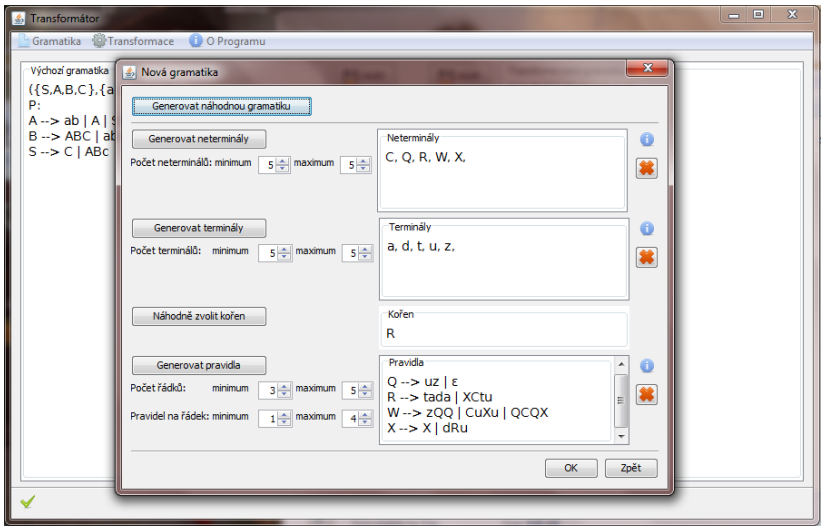


Obrázek 4.2: Screenshot výsledné verze programu

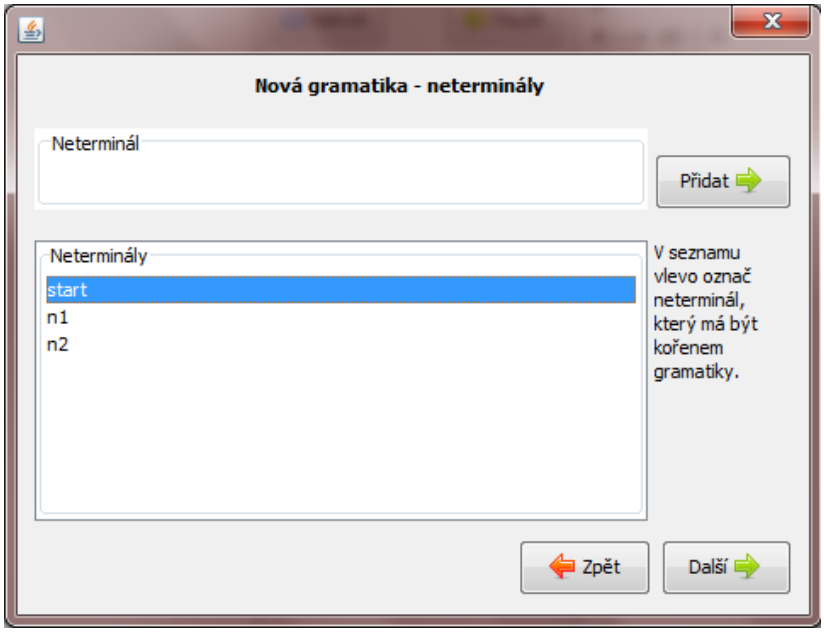
Finální verzi můžete vidět na obrázku 4.2. Jediné okno se dvěma výpisy — v levé části gramatika původní, v pravé gramatika po transformaci, případně posloupnost gramatik, je-li transformací provedeno několik, a pomocné množiny. Uprostřed mezi nimi rychle přístupné možnosti uložení, načtení gramatiky apod., včetně tří tlačítek s volitelnou funkcí.

Důležitou a dosud nepředstavenou částí GUI je vytvoření nové gramatiky. Původně bylo možné v aplikaci zadat gramatiku přímo do levého pole, bylo to řešení nevhodné a nyní je tento způsob znemožněn. Gramatiku je možné vytvořit pomocí *Menu* → *Nová gramatika* — zobrazí se okno se dvěma možnostmi. Nabízí se vytvoření nové gramatiky, tedy dle představ uživatele, a vygenerování gramatiky náhodné.

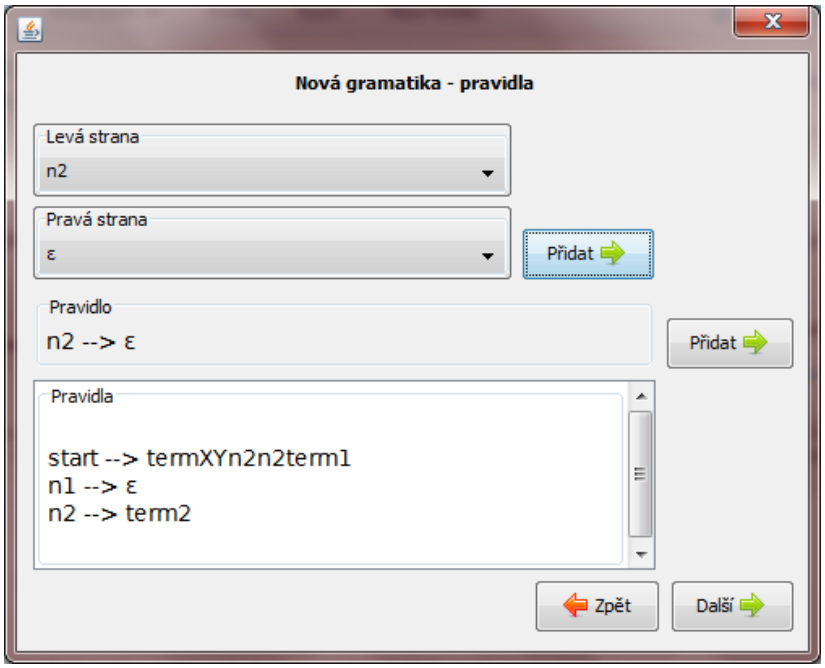
Při generování náhodné gramatiky uživatel pouze zadává několik informací typu maximální a minimální počet neterminálů, terminálů, pravidel. Je možné vygenerovat rovnou celou gramatiku, nebo také generovat po částech — pokud se mi například nebude líbit množina terminálů, vygeneruji ji znovu. Pro vzhled viz obrázek 4.3.



Obrázek 4.3: Screenshot okna pro generování náhodné gramatiky



Obrázek 4.4: Screenshot okna pro zadání neterminálů a kořene



Obrázek 4.5: Screenshot okna pro zadání pravidel

Zadávání nové gramatiky, jak ji chce uživatel, probíhá v několika po sobě jdoucích oknech, viz obrázky. Uživatel zadává neterminály a terminály zvlášť, v obdobně vypadajících oknech, při zadávání neterminálů je nutné označit kořen gramatiky 4.4. Pravidla se staví symbol po symbolu po vybrání neterminálu, který představuje levou stranu pravidla. Jakmile je pravidlo hotové, je třeba ho přidat 4.5. Posledním krokem už je jen přehled gramatiky v textovém formátu typickém pro tento program.

Nejoriginálnějším a z pohledu uživatele velice užitečným prvkem GUI jsou tři tlačítka ve středové části okna s konfigurovatelnou funkcionalitou. Uživatel si může sám vybrat, které tři transformace bude používat nejčastěji, a podle toho tlačítka nastavit. Prostřednictvím pravého tlačítka myši lze vyvolat menu s nabídkou transformací, které lze pro tlačítko vybrat. Změní-li se uživatelovy preference, jednoduše stejným způsobem prováděné transformace změní.

Nastavení těchto tří tlačítek si navíc program pamatuje, takže není nutné je při každém spuštění programu nastavovat znovu. Aktuální konfigurace je uložena v domovském adresáři uživatele v souboru s názvem *.transformator_gramatik.conf*.

4.2 Další možnosti vývoje

Pro tuto bakalářskou práci vytyčené cíle byly sice v programu splněny, existují ovšem další možnosti vývoje. Zejména je třeba doplnit nápovědu k programu, přestože je, doufám, natolik jednoduchý a snadno pochopitelný, že se bez ní uživatel obejde. Mám v úmyslu v tomto projektu pokračovat, pokud se ujme. Některé úkoly do budoucna jsou popsány níže.

Další transformace

Stále zbývá pár transformací bezkontextových gramatik, které tato aplikace neumožňuje realizovat. Jedná se o transformace probírané v rámci navazujícího kurzu k formálním automatům a gramatikám na FI MU. Syntaktické analyzátory nepracují s jakoukoli myslitelnou bezkontextovou gramatikou — vyžadují specifické typy, LL(k), SLL, LR q-gramatiky nebo LALR.

Především zbývá doplnit již načatou transformaci na LL(1) gramatiku a dopsat *pohlcení terminálního symbolu, resp. řetězce* a následně přidat propojení potřebných jednodušších transformací do kompletní transformace na LL(1) gramatiku. Obdobné kroky bude třeba provést pro LR a LALR gramatiky. Zatím nejtěžším úkolem se v tomto směru zdá nalezení přesných popisů algoritmů nebo alespoň jejich kvalitní slovní popis — včetně celé transformace z běžné bezkontextové gramatiky.

Konverze formátů

V současnosti program umožňuje ukládání gramatik v několika různých formátech. Mohlo by být užitečné umožnit i konverzi již uložené gramatiky z jednoho formátu do jiného. Za tímto účelem předpokládám vytvoření samostatného jednoduchého nástroje, který by

k takové konverzi mezi formáty využíval XML Stylesheet a XSLT, transformaci mezi XML formáty, a dále překlad mezi jinými formáty s pomocí vlastních gramatik pro tyto formáty.

Tento nástroj pak může sloužit samostatně pro konverzi souborů, jak si bude uživatel přát, nebo i ve spojení s již existující hlavní aplikací pro umožnění dosud neimplementovaného načítání gramatik i z jiných formátů než jen textového a XML.

Rozhranní pro terminál, příkazovou řádku

Někteří uživatelé preferují práci v terminálu a ovládání programu pomocí klávesnice a psaných příkazů před klikáním tlačítkem myši, nebo jen ocení, když se kvůli několika drobným výpočtům nemusí načítat rozsáhlé uživatelské rozhraní.

To mě vedlo k rozhodnutí modifikovat stávající aplikaci tak, aby bylo možné ji zpustit i přímo v terminálu s textovým uživatelským rozhraním. Program potom bude možné spouštět i s dalšími argumenty — vstupní a výstupní soubor a transformace, která se má provést. Tato možnost jen provede požadovanou transformaci na gramatice ze vstupního souboru, gramatiku uloží do výstupního souboru a program skončí. Mohlo by být užitečné pro jediný argument, název souboru, vygenerovat náhodnou gramatiku.

Bude třeba implementovat i výpis nápovědy k programu v této čistě textové verzi při zadání *program -h* nebo *program /?*. Stejná nápověda by měla být zobrazitelná i z textového rozhraní.

Program v textovém režimu bude umět gramatiky transformovat (tak jako grafická verze), načítat je ze souboru a ukládat. Stejně tak zůstane generování náhodné gramatiky a zadávání nové gramatiky, jen způsob bude upraven pro textové rozhraní. Bude tedy nutné analyzovat uživatelské příkazy ze standardního vstupu, filtrovat chyby, nechtěné i úmyslné, a předně navrhnout vhodný tvar příkazů a jejich slovník. Samotné transformace gramatik, ukládání a načítání souborů už jsou hotové a budou použity znovu.

Detailní krokování transformací

Vezmeme-li v úvahu, že tento program má být hlavně učební pomůckou, možnost projít si transformaci krok po kroku by byla docela uži-

tečná. Program zatím dokáže vypsat posloupnost výsledků transformací, je-li třeba jich provést víc najednou, a vypisuje pomocné množiny. Novou funkcí by bylo zobrazení postupných úprav gramatiky tak, jak jsou změny v průběhu transformace prováděny.

Implementace by se mohla ukázat jako lehce problematická, už jen protože program transformaci nedokáže provést přesně tak jako student, který má díky lidskému mozku přece jen poněkud odlišný pohled na problém. Přesto, každá transformace využívá cyklů a iterativního procházení, je tedy velice pravděpodobné, že drobným zásahem do kódu bude možné krokování transformací realizovat.

Bude třeba rozhodnout, zda se zobrazí rovnou všechny kroky, jak šly po sobě, nebo jestli uživatele radši program nechá potvrzovat každý krok, než zobrazí další. Případně by bylo možné toto rozhodování nechat na uživateli a přidat do programu další položku, kterou bude možno nastavit. Nehledě na zvolený způsob, bude třeba pozměnit návratovou hodnotu jednotlivých transformací z původní třídy Gramatika pravděpodobně na dvojici hodnot, třídy Gramatika a String, kde by byla v druhé hodnotě uchovávána posloupnost změn.

GUI a zvýraznění gramatiky pomocí HTML značek

V začátcích vývoje grafického rozhraní aplikace jsem počítala pouze s jednoduchým výpisem transformované gramatiky. Když pak v programu přibyla možnost zobrazit všechny provedené transformace, pokud jich bylo třeba provést několik najednou, zdá se prostý výpis možná poněkud nepřehledný. Jednotlivé transformace jsou sice oddělené, šlo by to však o něco zpřehlednit, pokud původní plochu pro výstup, komponentu JTextArea, nahradím komponentou JTextPane. Ta, narozdíl od JTextArea, nabízející pouze obyčejný text, umožňuje opatřit text HTML značkami, je tedy možné mít část textu tučně či kurzívou, podtržené apod.

Změna vyžaduje nepříjemný zásah do grafické složky programu, byla tedy zatím odložena na neurčito. Jisté je, že ke změně dojde a to zároveň s komplexním úklidem zdrojového souboru, který se o tuto část stará. Spolu s tímto dojde i k rozdělení souboru na menší, snadněji spravovatelné celky, bude proto nutné vyhradit dostatek času na provedení a opětovné testování funkčnosti.

Závěr

Cílem této práce bylo vytvořit grafickou desktopovou aplikaci, která by umožňovala uživateli provádět různé transformace bezkontextových gramatik zmiňované v předmětech *IB005 Formální jazyky a automaty I* a *IB102 Automaty a gramatiky*. Výsledná aplikace tohoto cíle dosáhla a nabízí i několik užitečných fukncí navíc. Jak zmiňuji v předchozí kapitole, i přesto je stále co přidávat či zlepšovat. Změny v GUI, krom zvýraznění výpisu gramatiky, nepředpokládám, bude však užitečné přidat i textové rozhraní pro práci v terminálu. Přibudou další nalezené transformace a k původnímu exportu výsledné gramatiky vznikne i jednoduchý nástroj pro konverzi formátů gramatik. Dále bude aplikace rozšířena o detailní krokování transformací.

Krom těchto plánovaných změn by se do budoucna dalo přemýšlet o rozšíření aplikace o zásobníkové automaty — ty s bezkontextovými gramatikami úzce souvisí, zvlášť vezmeme-li v úvahu praktické využití gramatik, tolikrát zmiňovanou syntaktickou analýzu. Nebylo by cílem vytváření vlastního plnohodnotného syntaktického analyzátoru ze zadané gramatiky jako spíš přidání další pomůcky k učení. Obdobně by mohl vzniknout například nástroj pro práci s regulárními jazyky a gramatikami a konečnými automaty.

Protože je program vyvíjen pod licencí GNU GPL, může si i kdokoli jiný vytvářet vlastní verze tohoto programu, přidávat vlastní funkcionalitu, nebo naopak odstranit si ze své osobní kopie to, co nepotřebuje. Kdokoli se může podílet na vývoji.

Práce na tomto projektu mi umožnila hlavně lépe ocenit úsilí, které vložili jiní do vývoje spousty dalších užitečných aplikací. Prohloubila jsem si znalosti a schopnosti programování v jazyce Java a mohla jsem se prakticky zabývat velice zajímavou částí informatiky. Celou práci hodnotím pozitivně, jako cennou zkušenost.

Literatura

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

[2] MATEESCU, Alexandru; SALOMAA, Arto. *Formal Languages: an Introduction and a Synopsis* [online]. In ROZENBERG, Grzegorz; SALOMAA, Arto. *Handbook of Formal Languages: Word, language, grammar* Springer, 1997 [cit. 8. 5. 2011]. Dostupné z WWW: <http://books.google.cz/books?id=yQ59ojndUt4C&lpg=PP1&pg=PP1#v=onepage&q&f=false>.

[3] Přispěvatelé Wikipedie. *Alan Turing* [online]. Wikipedie: Otevřená encyklopedie, c2011. Datum poslední revize 31. 03. 2011, [cit. 8. 05. 2011]. Dostupné z WWW: <http://cs.wikipedia.org/w/index.php?title=Alan_Turing&oldid=6717580>

[4] Přispěvatelé Wikipedie. *Turingův stroj* [online]. Wikipedie: Otevřená encyklopedie, c2011. Datum poslední revize 31. 03. 2011, [cit. 8. 05. 2011]. Dostupné z WWW: <http://cs.wikipedia.org/w/index.php?title=Turing%C5%AFv_stroj&oldid=6716718>

[5] Přispěvatelé Wikipedie. *L-systém* [online]. Wikipedie: Otevřená encyklopedie, c2011. Datum poslední revize 4. 04. 2011, [cit. 8. 05. 2011]. Dostupné z WWW: <http://cs.wikipedia.org/w/index.php?title=L-syst%C3%A9m&oldid=6731205>

[6] Přispěvatelé Wikipedie. *Noam Chomsky* [online]. Wikipedie: Otevřená encyklopedie, c2011. Datum poslední revize 20. 04. 2011, [cit. 8. 05. 2011]. Dostupné z WWW: <http://cs.wikipedia.org/w/index.php?title=Noam_Chomsky&oldid=6798616>

[7] Wikipedia contributors. *Automata theory* [online]. Wikipedia, The Free Encyclopedia, c2011. Datum poslední revize 3. 05. 2011, [cit. 9. 05. 2011]. Dostupné z WWW: <http://en.wikipedia.org/w/index.php?title=Automata_theory&oldid=427295751>

[8] ČERNÁ, Ivana; KŘETÍNSKÝ, Mojmír; KUČERA, Antonín. *Formální jazyky a automaty I* [online]. Elportál, Brno : Masarykova univerzita, 2006 [cit. 8. 05. 2011]. Dostupné z WWW: <http://is.muni.cz/elportal/?id=703389>

[9] Dr. Hashim Habiballa, PhD. *Regulární a bezkontextové jazyky II* [online]. Ostravská univerzita v Ostravě, 2005 [cit. 13. 05. 2011]. Dostupné z WWW: <http://www1.osu.cz/home/habibal/publ/rabj2.pdf>

[10] Přispěvatelé Wikipedie. *Chomského hierarchie* [online]. Wikipedie: Otevřená encyklopedie, c2011. Datum poslední revize 6. 01. 2011, [cit. 8. 05. 2011]. Dostupné z WWW: <http://cs.wikipedia.org/w/index.php?title=Chomsk%C3%A9ho_hierarchie&oldid=6307470>

[11] Wikipedia contributors. *Yacc* [online]. Wikipedia, The Free Encyclopedia, c2011. Datum poslední revize 26. 04. 2011, [cit. 10. 05. 2011]. Dostupné z WWW: <http://en.wikipedia.org/w/index.php?title=Yacc&oldid=426008196>

[12] Wikipedia contributors. *GNU bison* [online]. Wikipedia, The Free Encyclopedia, c2011. Datum poslední revize 27. 04. 2011, [cit. 10. 05. 2011]. Dostupné z WWW: <http://en.wikipedia.org/w/index.php?title=GNU_bison&oldid=426200259>

[13] Wikipedia contributors. *JavaCC* [online]. Wikipedia, The Free Encyclopedia, c2011. Datum poslední revize 25. 01. 2011, [cit. 10. 05. 2011]. Dostupné z WWW:
<http://en.wikipedia.org/w/index.php?title=JavaCC&oldid=409870393>

[14] Wikipedia contributors. *Backus–Naur Form* [online]. Wikipedia, The Free Encyclopedia, c2011. Datum poslední revize 13. 05. 2011, [cit. 16. 05. 2011]. Dostupné z WWW:
<http://en.wikipedia.org/w/index.php?title=Backus%E2%80%93Naur_Form&oldid=428864710>

[15] Refsnes Data. *DTD Tutorial* [online]. [cit. 10. 05. 2011]. Dostupné z WWW:
<http://www.w3schools.com/dtd/>

[16] Free Software Foundation, Inc. *Bison - GNU parser generator* [online]. Datum poslední revize 5. 01. 2009, [cit. 11. 05. 2011]. Dostupné z WWW:
<http://www.gnu.org/software/bison/>

[17] Přispěvatelé Wikipedie. *Transformace na LL(1)* [online]. Wikipedie: Otevřená encyklopedie, c2011. Datum poslední revize 12. 06. 2010, [cit. 13. 05. 2011]. Dostupné z WWW:
<http://cs.wikipedia.org/w/index.php?title=Transformace_na_LL(1)&oldid=5458346>

[18] Oracle. *www.java.com* [online]. [cit. 13. 05. 2011].
Dostupné z WWW:
<http://www.java.com>

[19] Free Software Foundation, Inc. *GNU GENERAL PUBLIC LICENSE* [online]. Version 3, 29 June 2007. Datum poslední revize 12. 06. 2010, [cit. 16. 05. 2011]. Dostupné z WWW:
<http://www.gnu.org/licenses/gpl.html>