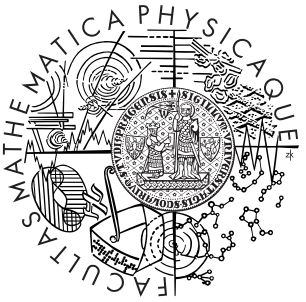
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Martina Krejčová

Nástroj pro testování algoritmů pro učení jazyků

Kabinet software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Petr Hoffmann
Studijní program: Informatika, programování

2009

Děkuji RNDr. Petru Hoffmannovi za ochotu při konzultacích a cenné rady, které mi poskytl jako vedoucí mé bakalářské práce.

Prohlašuji, že jsem svou bakalářskou práci napsala samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne Martina Krejčová

Obsah

1 Úvod **7**
1.1 Jazyk . . . 8
1.2 Gramatika . . . 8
1.2.1 Generování slov jazyka . . . 8
1.3 Bezkontextová gramatika . . . 9
1.3.1 Chomského hierarchie . . . 9
1.3.2 Chomského normální forma . . . 10
1.3.3 Zásobníkový automat . . . 10
1.3.4 Krok zásobníkového automatu . . . 11
1.3.5 Jazyk zásobníkového automatu . . . 11
1.3.6 Deterministický zásobníkový automat . . . 12
1.3.7 Ekvivalence gramatik . . . 13
1.4 Gramatická inference . . . 13

2 Testování algoritmů **15**
2.1 Omphalos . . . 15
2.2 Vstupní data . . . 16
2.3 Obtížnost gramatiky . . . 16
2.4 Trénovací a testovací data . . . 24
2.4.1 Postup generování dat . . . 24
2.4.2 Vylepšení generování testovacích dat . . . 25
2.5 Generování gramatiky . . . 25
2.6 Ohodnocení algoritmů a srovnání jejich úspěšnosti . . . 26
2.6.1 Procentuální úspěšnost . . . 26
2.6.2 Ratio . . . 26
2.6.3 F-measure . . . 27

3 Závěr **28**

4 Uživatelská příručka **30**
4.1 Instalace 30
4.2 Vytváření gramatiky 31
4.2.1 Generování gramatiky 31
4.2.2 Vytváření vlastní gramatiky 31
4.3 Generování trénovacích a testovacích dat 32
4.3.1 Formát dat 32
4.4 Provedení jednoho testu 32
4.5 Provedení velkého testu 33
4.6 Algoritmy vhodné pro testování 33
4.7 Příklad použití 34
4.7.1 Gramatika 34
4.7.2 Trénovací a testovací data 35
4.7.3 Testování algoritmů 36

5 Programátorská dokumentace **41**
5.1 Popis tříd a pomocných funkcí 41
5.2 Okna a dialogy 42
5.3 Algoritmy k testování 43
5.4 Požadavky 43

Literatura **44**

Název práce: Nástroj pro testování algoritmů pro učení jazyků
Autor: Martina Krejčová
Katedra (ústav): Kabinet software a výuky informatiky
Vedoucí bakalářské práce: RNDr. Petr Hoffmann
e-mail vedoucího: petr.hoffmann@mff.cuni.cz

Abstrakt: Úkolem gramatické inference je nalezení pravidelností v datech. Je-li pro data z těchto odpozorovaných pravidel vytvořen model, můžeme pomocí tohoto modelu data například zkomprimovat, vytvořit data nová, která budou tato pravidla také splňovat, či můžeme určit, která data tomuto modelu odpovídají a která ne. Cílem této práce bylo vytvořit testovací prostředí pro algoritmy gramatické inference bezkontextových jazyků. Model dat v našem prostředí tvoří bezkontextové gramatiky. Je zde popsán postup generování gramatik a následně generování dat, z kterých se algoritmus může pokusit bezkontextový jazyk naučit. Za účelem srovnání úspěšnosti algoritmu na datech generovaných z více různých bezkontextových gramatik je řešen pojem složitosti bezkontextových gramatik. Dále je navrženo několik variant vyhodnocení úspěšnosti algoritmů.

Klíčová slova: gramatická inference, bezkontextový jazyk, učení

Title: Tool for testing languages learning algorithms
Author: Martina Krejčová
Department: Departement of Software and Computer Science Education
Supervisor: RNDr. Petr Hoffmann
Supervisor's e-mail address: petr.hoffmann@mff.cuni.cz

Abstract: Goal of this work was to develop a tool for testing algorithms of grammatical inference of context-free languages. Grammatical inference is a process of learning of grammars and languages from data. Learning could mean finding a suitable model that describes data. Due to this model we could for instance compress this data, create new data or find out which data are consistent with this model. The model in our tool is context-free grammar. We describe how to generate the context-free grammar and data from which the algorithm can try to learn the language of this grammar. The problem of context-free grammar complexity was solved in order to evaluate success achieved by an algorithm in different learning tasks. Also some alternatives of evaluating success of an algorithm are described. This tool is also useful to create data for publishing results of the algorithm.

Keywords: grammatical inference, context-free language, learning

Kapitola 1

Úvod

Vyvíjíme-li nějaký nástroj, vždy se nám hodí vědět, zda právě jdeme správnou cestou, jestli jsme danými úpravami nástroj vylepšili, zda je tento nástroj již vhodný k použití nebo je na něm nutné dále pracovat. Potřebujeme tedy nástroj otestovat.
Naším cílem v této práci je vytvoření nástroje, který umožní porovnání výkonu algoritmů gramatické inference bezkontextových jazyků. Stručně lze říci, že úkolem algoritmů gramatické inference je odpozorování pravidel, kterými se řídí data předložená algoritmu. Tyto znalosti lze pak použít například k rozeznání dat, která se těmito pravidly také řídí.
Gramatická inference se používá v počítačovém zpracování přirozeného jazyka, kompresi dat, tvoření přizpůsobivých inteligentních agentů, určování diagnóz, v biologii, data miningu či rozpoznávání hudebních stylů a generování nových melodií.
Porovnání schopností může proběhnout buď u několika různých algoritmů navzájem či jednoho algoritmu při různé složitosti zadání. Při testování se může uživateli hodit i srovnání s algoritmy regulární inference, které jsme v našem testovacím nástroji umožnili použít. Tyto algoritmy byly vzaty ze soutěže Abbadingo [7].
První kapitola této práce je seznámení s teoretickým základem práce. Jsou zde definovány pojmy a zmíněny věty z oblasti bezkontextových gramatik, které budeme dále používat. Kapitola 2 se zabývá již testováním algoritmů. Je zde zmíněna soutěž v gramatické inferenci Omphalos [3], z jejíchž materiálů bylo hodně čerpáno. Dále je zde popsáno vytvoření testovací úlohy, spočítání její obtížnosti, způsob zadání této úlohy testovaným algoritmům a následné vyhodnocení úspěšnosti algoritmů. Kapitola 3 shrnuje výsledky této práce. Kapitola 4 je uživatelská dokumentace, kapitola 5 programátorská dokumentace k přiložené aplikaci, která implementuje postupy popsané v této práci.

Teorie popisovaná v této kapitole je použita ze zdrojů [4] a [5].

1.1 Jazyk

Chceme-li se dále zabývat teorií formálních jazyků, musíme si říci, co vlastně jazyk je a vysvětlit si další pojmy, které budeme dále používat.

Definice 1 *Abeceda je konečná, neprázdná množina symbolů, kterou budeme značit* Σ.

Definice 2 *Řetězec nebo slovo je konečná sekvence symbolů vybraných z nějaké abecedy.*
Množina všech řetězců nad abecedou Σ *se značí* Σ^*.

Definice 3 *Speciálním případem je prázdné slovo, což je posloupnost znaků nulové délky. Budeme ho značit* ϵ.

Definice 4 $L \subseteq \Sigma^*$ *se nazývá jazyk.*

1.2 Gramatika

K popisu jazyka lze využít různých prostředků, jedním z nich je gramatika. Gramatika je popis jazyka pomocí pravidel, podle kterých se vytvářejí řetězce daného jazyka.

Definice 5 *Gramatika je čtveřice* $G(\Gamma, \Sigma, S, \Pi)$*, kde* Γ *je konečná množina neterminálních symbolů,* Σ *konečná množina terminálních symbolů taková, že* $\Gamma \cap \Sigma = \emptyset, S \in \Gamma$ *počáteční neterminál,* Π *systém produkcí* $u \rightarrow v$*, kde* $u, v \in (\Gamma \cup \Sigma)^*$.

1.2.1 Generování slov jazyka

Generování slov je iterativní proces, začínáme s řetězcem, který je složen jen z počátečního neterminálu. V každém kroku vždy náhodně vybereme z našeho řetězce část, která se nachází na levé straně některého z pravidel. Vybranou část řetězce přepíšeme pravou stranou tohoto pravidla. Tento proces končí, když je řetězec tvořen pouze z terminálních symbolů. Takto vytvořený řetězec je slovem z jazyka dané gramatiky.

Definice 6 *Nechť G je gramatika $G(\Gamma, \Sigma, S, \Pi)$, $\alpha, \beta, \gamma \in (\Sigma \cup \Gamma)^*$, $A \in (\Sigma \cup \Gamma)^+$. Nechť $A \to \gamma$ je pravidlo z Π. Potom říkáme, že $\alpha A \beta \implies \alpha\gamma\beta$. Říkáme, že w se přepíše na z ($w \implies^* z$) $w, z \in (\Sigma \cup \Gamma)^*$, jestliže existuje $u_1 ... u_n \in (\Sigma \cup \Gamma)^*$ takové, že $w = u_1 \implies u_2 \implies ... \implies u_n = z$. Posloupnost $u_1, ..., u_n$ nazýváme derivací.*

Definice 7 *Jazyk $L(G)$ generovaný gramatikou G definujeme jako množinu všech terminálních řetězců, které lze odvodit z počátečního neterminálního symbolu. Značíme $L(G) = \{w | w \in \Sigma^*, S \implies^* w\}$.*

K zachycení odvození slova lze použít kromě derivace i tzv. derivační strom.

Definice 8 *Derivační stromy pro gramatiku $G(\Gamma, \Sigma, S, \Pi)$ jsou stromy s následujícími vlastnostmi.*

1. *Každý vnitřní uzel je označen neterminálem z Γ.*
2. *Každý list je označen buď neterminálem z Γ, terminálem z Σ nebo ϵ. Pokud je list označen ϵ, potom musí být jediným synem svého otce.*
3. *Pokud je vnitřní uzel označen A a jeho synové jsou zleva označeni $X_1, X_2, ..., X_k$, potom $A \to X_1 X_2 ... X_k$ je pravidlo z Π.*

1.3 Bezkontextová gramatika

1.3.1 Chomského hierarchie

V teorii formálních jazyků se ukázalo, že kladením různých omezení na tvar přepisovacích pravidel gramatik lze omezit jejich vyjadřovací sílu. Základním dělením je tzv. Chomského hierarchie.

Definice 9 Gramatika typu 0 *je gramatika s pravidly v obecném tvaru, generuje rekurzivně spočetné jazyky.*

Gramatika typu 1 *je gramatika s pravidly pouze ve tvaru $\alpha X \beta \to \alpha w \beta$, kde $X \in \Gamma, \alpha, \beta \in (\Gamma \cup \Sigma)^*, w \in (\Gamma \cup \Sigma)^+$. Výjimkou je pravidlo $S \to \epsilon$, kde S je počáteční neterminál a ϵ je prázdné slovo. Potom se ale S nevyskytuje na pravé straně žádného pravidla. Gramatiky typu 1 generují kontextové jazyky.*

Gramatika typu 2 *je gramatika s přepisovacími pravidla ve tvaru* $X \to w$, *kde* $X \in \Gamma, w \in (\Gamma \cup \Sigma)^*$. *Generují bezkontextové jazyky.*

Gramatika typu 3 *jsou gramatiky s přepisovacími pravidly pouze ve tvaru* $X \to wY, X \to w$, *kde* $X, Y \in \Gamma, w \in \Sigma^*$. *Jazyky, které generují, se nazývají regulární jazyky.*

Jazyky z Chomského hierarchie jsou uspořádány od nejobecnějších po nejspecifičtější. Specifičtější jsou vlastní podmnožinou obecnějších.

Věta 1 *rekurzivně spočetné jazyky* $\supset$ *kontextové jazyky* $\supset$ *bezkontextové jazyky* $\supset$ *regulární jazyky*

Cílem této práce je testovat algoritmy inference bezkontextových gramatik. Lze však testovat algoritmy určené pro inferenci jazyků generovaných gramatikami všech typů. Jen je třeba počítat s tím, že algoritmy regulární inference nemusí dosáhnout tak dobrých výsledků.

1.3.2 Chomského normální forma

Někdy se nám může hodit, aby pravidla gramatiky byla ve speciálním tvaru. Následující tvar má tu výhodu, že derivační strom slova odvozeného z těchto pravidel je skoro binární. Díky tomu se dá odhadovat hloubka stromu pro slovo dané délky, kde hloubka je vlastně počet derivací, kterými musíme projít, abychom ze startovacího neterminálu odvodili terminální slovo.

Definice 10 *Říkáme, že gramatika je v Chomského normální formě, jestliže všechna její pravidla mají tvar* $X \to YZ$ *nebo* $X \to a$, *kde* $a \in \Sigma, X, Y, Z \in \Gamma$.

Věta 2 *Ke každému bezkontextovému jazyku* L *existuje bezkontextová gramatika* G *v Chomského normální formě taková, že* $L(G) = L \setminus \{\epsilon\}$.

1.3.3 Zásobníkový automat

Alternativním popisem bezkontextového jazyka ke gramatice je zásobníkový automat. Na rozdíl od gramatiky slova negeneruje, slova jsou automatu předkládána a ten rozhoduje, zda patří do jazyka.

Definice 11 *Zásobníkový automat je složen ze sedmi komponent. Zásobníkový automat P zapisujeme $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$.*
Kde :
Q *je konečná množina stavů,*
Σ *je množina vstupních symbolů,*
Γ *je konečná abeceda zásobníku,*
δ *je přechodová funkce* $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to Q \times \Gamma^*$,
q_0 *je počáteční stav zásobníkového automatu,* $q_0 \in Q$
Z_0 *je startovací symbol na zásobníku,* $Z_0 \in \Gamma$
F *je množina koncových stavů,* $F \subseteq Q$.

1.3.4 Krok zásobníkového automatu

Definice 12 *Konfigurace zásobníkového automatu je popsána trojicí* (q, w, γ), *kde*
q *je stav zásobníkového automatu,*
w *je zbývající vstup a*
γ *je obsah zásobníku.*
Vrchol zásobníku je umístěn na levé straně γ *a dno zásobníku je na pravém konci* γ.

Krok zásobníkového automatu spočívá v přechodu mezi jednotlivými konfiguracemi. Pokud má zásobníkový automat konfiguraci $(q, aw, X\beta)$ a $(p, \alpha) \in \delta(q, a, X)$, může odebráním a (a může být i ϵ) ze vstupu a nahrazením X na vrcholu zásobníku za α přejít do konfigurace $(p, w, \alpha\beta)$.

Definice 13 *Nechť* $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ *je zásobníkový automat, potom krok zásobníkového automatu* $\vdash$ *definujeme následovně. Pokud* $(p, \alpha) \in \delta(q, a, X)$ *, potom pro každý řetězec* $w \in \Sigma^*$ *a* $\beta \in \Gamma^*$*:*
$(q, aw, X\beta) \vdash (p, w, \alpha\beta)$.

Pro znázornění nula nebo více kroků zásobníkového automatu se používá $\vdash^*$.

1.3.5 Jazyk zásobníkového automatu

Jak jsme již řekli, zásobníkový automat pracuje tak, že dostane na vstupu slovo a rozhodne, zda jej přijme nebo ne. Nyní budeme definovat, co znamená, že automat přijímá dané slovo nebo jazyk.

Zásobníkový automat může přijímat vstup buď koncovým stavem nebo prázdným zásobníkem. Tyto dvě metody jsou ekvivalentní. Pro jazyk L existuje zásobníkový automat, který ho přijímá prázdným zásobníkem právě tehdy, když pro jazyk L existuje zásobníkový automat, který ho přijímá koncovým stavem.

Definice 14 *Nechť* $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ *je zásobníkový automat. Potom* $L(P) = \{w|(q_0, w, Z_0) \vdash {}^*(q, \epsilon, \alpha)\, q \in F, \alpha \in \Gamma^*, w \in \Sigma^*\}$ *je jazyk přijímaný P koncovým stavem.*

Tedy P začíná v počátečním stavu s inicializovaným zásobníkem s w čekajícím na vstupu, P postupně odebere w ze vstupu a dostane se do přijímacího stavu. Obsah zásobníku na konci není důležitý.

Definice 15 *Pro každý zásobníkový automat* $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ *je také definována množina slov* $N(P) = \{w|(q_0, w, Z_0) \vdash {}^*(q, \epsilon, \epsilon), q \in Q, w \in \Sigma^*\}$. *N(P) je množina vstupů w, které P odebere ze vstupu a ve stejném čase vyprázdní svůj zásobník. Tato slova nazýváme přijímaná prázdným zásobníkem.*

Věta 3 *Jazyky přijmané zásobníkovým automatem koncovým stavem nebo prázdným zásobníkem jsou právě bezkontextové jazyky.*

1.3.6 Deterministický zásobníkový automat

Existuje omezená varianta zásobníkového automatu, který má při každé konfiguraci jevýše jednu možnost kroku. Tomuto automatu se říká deterministický zásobníkový automat. Tato vlastnost zásobníkovému automatu snižuje jeho sílu a jazyky jím přijímané jsou podmnožinou jazyků bezkontextových.

Definice 16 *Zásobníkový automat* $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ *je deterministický, pokud jsou splněny následující podmínky.*

1. *pro každé* $q \in Q, a \in (\Sigma \cup \{\epsilon\}), X \in \Gamma$ *obsahuje* $\delta(q, a, X)$ *nejvýše jeden prvek.*

2. *Pokud pro nějaké* $a \in \Sigma$, $\delta(q, a, X) \neq \emptyset$, *potom* $\delta(q, \epsilon, X) = \emptyset$.

Definice 17 *Jazyky přijmané deterministickým zásobníkovým automatem koncovým stavem nebo prázdným zásobníkem jsou deterministické bezkontextové jazyky.*

Definice 18 *Deterministické bezkontextové gramatiky jsou takové gramatiky, jejichž jazyk může být přijmán deterministickým zásobníkovým automatem koncovým stavem nebo prázdným zásobníkem.*

1.3.7 Ekvivalence gramatik

Jedním z našich cílů při testování algoritmů gramatické inference je určení úspěšnosti algoritmu. K tomuto účelu by se nám hodilo mít možnost zjistit, zda algoritmus odhalil hledanou námi vygenerovanou gramatiku. V této části se dozvíme, že tento problém nemůžeme algoritmicky vyřešit a proč tedy budeme muset v následující kapitole zvolit pro určování úspěšnosti jinou metodu, která sice nebude úplně přesná, ale dá se implementovat.

Definice 19 *Gramatiky G_1 a G_2 jsou ekvivalentní pokud $L(G_1) = L(G_2)$.*

Věta 4 *Pro bezkontextové gramatiky G_1 a G_2 je algoritmicky nerozhodnutelné zda $L(G_1) = L(G_2)$.*

Dvě bezkontextové gramatiky jsou tedy ekvivalentní, pokud se rovnají jejich jazyky.
Podle předcházející věty je tedy ekvivalence dvou bezkontextových gramatik algoritmicky nerozhodnutelný problém.

1.4 Gramatická inference

Testovací prostředí popsané v této práci testuje algoritmy gramatické inference. V této části se dozvíme základní definice z této oblasti.
Gramatická inference je proces učení se gramatik a jazyků z dat. Jedná se o odpozorování, odvozování pravidel ze vstupních dat, která se těmito pravidly řídí. Znalost těchto pravidel nám následně umožní klasifikaci dat se kterými jsme se dosud nesetkali, zestručnění těchto dat a jejich popsání vhodným modelem. Takovýmto modelem může být gramatika.
Nejdříve si řekneme, co jsou vlastně trénovací data, která jsou vstupem pro algoritmy gramatické inference a co v našem případě od algoritmů čekáme za výstup.

Definice 20 *Nechť L je cílový jazyk. Pozitivní vzorek je S_+ konečná podmnožina L, negativní vzorek je S_- konečná podmnožina $\bar{L}$.*

Definice 21 *Trénovacími daty pro jazyk L nazveme $S = S_+ \cup S_-$.*

Vstupem algoritmů gramatické inference jsou trénovací data. Cílem algoritmů je naučit se z těchto trénovacích dat hledaný jazyk nebo se mu co nejvíce přiblížit. Řešení a výstup vydaný algoritmy gramatické inference mohou být různé, záleží na způsobu dalšího použití daného algoritmu. Někde může být vyžadován jako výstup model, který by měl být co nejmenší nebo

model, který může být i větší, ale rychleji získaný. V našem případě je úkolem algoritmu úspěšná klasifikace dat, s kterými se ještě nesetkal. Očekávaným výstupem je tedy soubor s oklasifikovanými daty, která jsou algoritmu předána.

Kapitola 2

Testování algoritmů

V této kapitole se dostáváme k samotnému testování algoritmů gramatické inference. Seznámíme se se soutěží Omphalos a jejími postupy, kterými byla tato práce inspirována. Je zde popsán postup vytváření úlohy, postup generování dat pro učení se algoritmů a dat pro jejich testování. Na konci kapitoly jsou uvedeny způsoby vyhodnocení úspěšnosti algoritmů, které jsme použili. V této kapitole bylo čerpáno z internetových stránek soutěže Omphalos [2] a z materiálu [1], který nám byl poskytnut od tvůrců této soutěže.

2.1 Omphalos

Testovací prostředí v této práci je inspirováno soutěží Omphalos. Omphalos byla soutěž v učení bezkontextových jazyků. Úkolem bylo odvodit model bezkontextového jazyka z příkladů řetězců do jazyka patřících i nepatřících a tento model následně použít k označení množiny testovacích řetězců, zda jsou řetězci z daného jazyka nebo ne. Tato soutěž byla zorganizována po úspěchu soutěže Abbadingo v regulární inferenci, díky níž došlo v této oblasti k pokrokům.
V naší práci byl především převzat postup vytvoření gramatiky, výpočet její složitosti a způsob generování dat tak, aby co nejlépe vystihovaly jazyk dané gramatiky. Srovnání dovedností algoritmů jsme převzít nemohli, neboť v soutěži neprobíhalo. Za úspěch bylo pokládáno jen stoprocentně správné označení dat, jako do jazyka patřících nebo nepatřících. Průběžným vítězem byl vždy ten, kdo jako první vyřešil nejsložitější z vyřešených úloh.

2.2 Vstupní data

Algoritmům budou jako trénovací data poskytnuty řetězce označené, zda z daného jazyka jsou nebo ne. Pozitivní příklady jsou generovány z gramatiky, která je vytvořena buď automaticky nebo uživatelem.
Výhodou je snadná reprezentace trénovacích dat. Jde jen o řetězce s označením zda do daného jazyka patří či nepatří. Dalším kladem je v případě nekonečnosti jazyka možnost kdykoliv dogenerovat další data.
Pro bezkontextový jazyk často existuje více gramatik, které tento jazyk generují. Testovaný algoritmus tedy může dojít ke gramatice jiné, než je gramatika cílová i v případě, že se naučil daný jazyk. Ekvivalence dvou bezkontextových gramatik je algoritmicky nerozhodnutelný problém.
Přestože je problém ekvivalence gramatik algoritmicky nerozhodnutelný, chceme gramatiky porovnat. K tomu slouží testovací data. Sada testovacích dat je dalším vstupem algoritmu, tato data již nejsou označena, zda jsou z jazyka či ne. Toto označení je úkolem testovaného algoritmu. Ohodnocení algoritmu poté spočívá v určení jeho úspěšnosti při označení testovacích dat. Tím se zbavíme problému porovnávání původní gramatiky s tou, která byla vytvořena algoritmem.
Algoritmu však musí být poskytnuta dostatečně velká testovací data, aby bylo poznatelné, jak moc se jeho model jazyka podobá cílovému.

2.3 Obtížnost gramatiky

V této části popíšeme postup jakým jsou generována trénovací data tak, aby se testovaný algoritmus mohl co nejvíce přiblížit k cílové gramatice. Pomocí algoritmů je popsán teoretický postup, kterým bychom vygenerovali data, z kterých by se testovaný algoritmus mohl naučit přímo cílový jazyk. Avšak ne každý z uvedených algoritmů by bylo možné jednoduše implementovat. V našem řešení se tedy pouze tomuto postupu přiblížíme.
Schopnosti algoritmu budou pravděpodobně zjišťovány na více různých úlohách, proto je vhodné mít možnost určit obtížnost jednotlivých úloh. K tomu potřebujeme míru obtížnosti gramatik. Míra, kterou zde používáme, byla převzata ze soutěže Omphalos. Byla odvozena z modelu řešení gramatické inference v konečném čase hledáním hrubou silou, takzvaným BruteForceLearner algoritmem.
BruteForceLearner algoritmus je schopen rozeznat gramatiku z pozitivních a negativních dat v exponenciálním čase. Je známo, že bezkontextové gramatiky nemohou být rozeznány v konečném čase bez negativních příkladů. Je proto potřeba poskytnout i negativní data.

Velikost prostoru hypotéz vytvořeného BruteForceLearner algoritmem je použita jako míra obtížnosti úlohy. Tento model byl také použit k určení, zda jsou vstupní data dostatečná k rozeznání cílové gramatiky a zda je tedy možné úlohu vyřešit. Říká nám tedy i jakým způsobem mají být konstruována trénovací data.
Následující algoritmy popisují BruteForceLearner algoritmus a algoritmy pro vytvoření trénovacích dat tak, aby z nich bylo možné odvodit gramatiku popisující daný jazyk.

ConstructAllGrammars(O) dostane množinu slov O a vytvoří z nich gramatiky v Chomského normální formě. Tyto gramatiky jsou všechny možné, jejichž pravidla jsou podmnožinou pravidel v Chomského normální formě používajících terminály z řetězců z množiny O a neterminály z derivačních stromů D. D jsou všechny možné derivační stromy slov z O, které mají v každém vnitřním uzlu unikátní neterminál, kromě kořene, kde má každý strom stejný počáteční neterminál.

Algorithm 1 ConstructAllGrammars(O)

1. O = množina slov z trénovacích dat patřících do jazyka
2. $\Sigma = \{b|abc \in O, a, c \in \Sigma^*\}$ /*je množina všech terminálních symbolů v O*/
3. $N = \{N_\mathrm{i}|1 \leq i \leq ((\sum_j (2 \times |O_\mathrm{j}| - 2)) + 1)\} \cup \{S\}$ /*O_j jsou jednotlivá slova z množiny O*/
/*N tvoří množinu neterminálů*/
4. $\Pi = \{S \rightarrow \epsilon\} \cup \{A \rightarrow BC|A, B, C \in N\} \cup \{A \rightarrow a|A \in N, a \in \Sigma\}$ /*je sada možných pravidel používajících Σ a N*/
5. $G = (N, \Sigma, \Pi, S)$ je gramatika v Chomského normální formě, která obsahuje všechny možná pravidla používající Σ a N
6. $H = \{(N_\mathrm{i}, \Sigma, \Pi_\mathrm{i}, S)|\Pi_\mathrm{i} \subseteq \Pi\}$ /*jsou všechny možné gramatiky generující jazyky, které jsou podmnožinou jazyka generovaného gramatikou G*/
7. return H

PruneGrammars(O^+, O^-) z množiny řetězců pozitivních příkladů O^+, které dostane na vstupu, zkonstruuje pomocí algoritmu ConstructAllGrammars(O^+) hypotetické gramatiky. Z těchto gramatik následně odstraní ty, které by byly nekonzistentní se vstupními daty.

Algorithm 2 PruneGrammars(O^+, O^-)

1. /*O^+ je množina všech slov z trénovacích dat patřících do jazyka cílové gramatiky*/
   /*O^- je množina všech slov z trénovacích dat nepatřících do jazyka cílové gramatiky*/
2. $H = ConstructAllGrammars(O^+)$
3. smažou se všechny gramatiky z H nekonzistentní s O^+ nebo O^-
4. return H

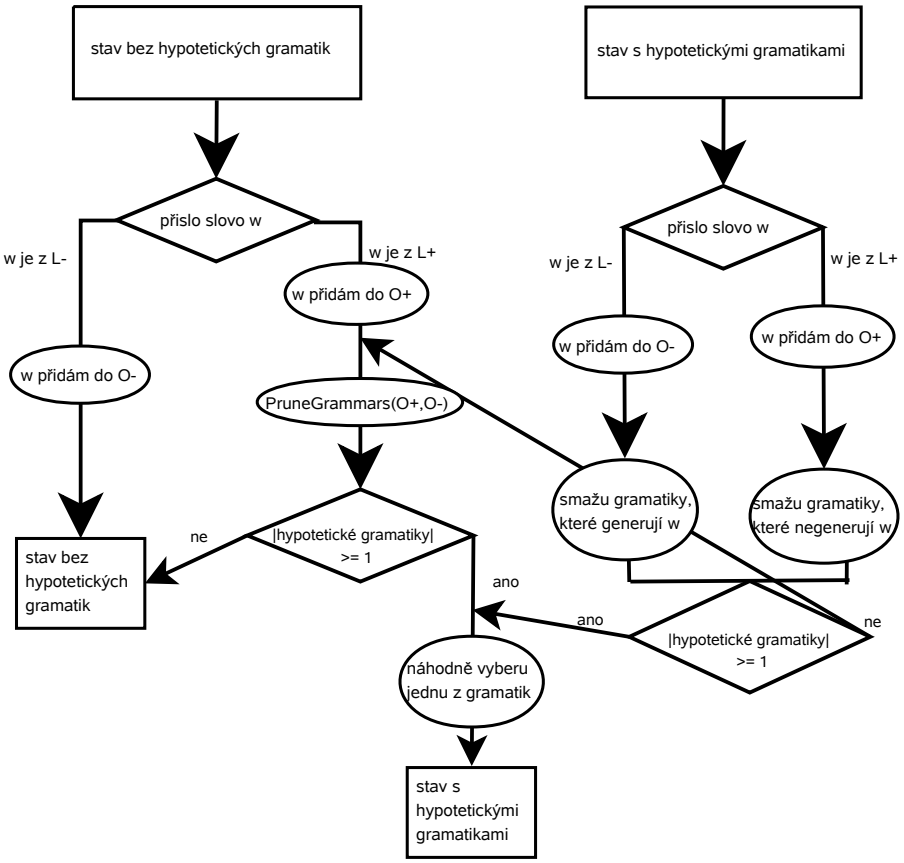
BruteForceLearner() algoritmus začíná s množinou gramatik H, která obsahuje jedinou gramatiku, a to generující pouze prázdné slovo. V každém cyklu náhodně vybere jednu z gramatik, označí ji jako možnou správnou gramatiku, načte další řetězec O_{n} a přidá ho do množiny pozitivních(O^+) nebo negativních(O^-) příkladů. Následně z H vyřadí všechny gramatiky, které by byly nekonzistentní s daty v O^+ nebo O^-. Pokud by po tomto kroku byla množina H prázdná, bude do H vygenerována nová množina gramatik konzistentních s daty v O^+ a O^-. Algoritmus BruteForceLearner je nekonečný, stále od nás očekává data a ta pak zpracovává. Ilustruje jakým způsobem se dá dojít k hledané gramatice. Jedna z následujících vět nám říká, že po určitém čase bude H v algoritmu obsahovat gramatiky, které jsou ekvivalentní s hledanou gramatikou.
Postup algoritmu je znázorněn na diagramu 2.1.

V dalším postupu budeme potřebovat pojem redundantního pravidla, hlavně tedy pojem gramatiky bez redundantních pravidel.

Definice 22 *Pravidlo $p \in \Pi$ gramatiky $G = (N, \Sigma, \Pi, S)$ je redundantní, pokud se jeho odstraněním z množiny pravidel Π nezmění jazyk generovaný gramatikou G. Tedy $p \in \Pi$ je redundantní pravidlo, pokud $G = (N, \Sigma, \Pi, S)$, $G' = (N, \Sigma, \Pi \setminus \{p\}, S)$ a L(G) = L(G').*

Lemma 1 *Nechť $G = (N, \Sigma, \Pi, S)$ je gramatika v Chomského normální formě bez redundantních pravidel.*

1. *CalcDerivations(G) vrací konečnou množinu derivací D, kde každá derivace d_i je unikátní,*
2. *$d_i =$ "$S \implies {}^*w$" $\in D$ odvozuje unikátní slovo w, tak že každé pravidlo z G je v D použito alespoň jednou.*



Obrázek 2.1: BruteForceLearner diagram

Algorithm 3 BruteForceLearner()

$H = \{(S, \emptyset, S \rightarrow \epsilon, S)\}$
loop
  náhodně vybereme jednu z gramatik z H jako hypotetickou gramatiku
  vezmeme další řetězec O_n
  if O_n je řetězec patřící do jazyka **then**
    $O^+ = O^+ \cup \{O_n\}$
  else
    $O^- = O^- \cup \{O_n\}$
  end if
  smažeme všechny gramatiky z H nekonzistentní s O^+ nebo O^-
  if $|H| = 0$ **then**
    $H = PruneGrammars(O^+, O^-)$
  end if
end loop

CalcObservations(G) Ke každému pravidlu gramatiky G najde slovo, kterého každý derivační strom používá dané pravidlo. Tedy takové slovo, které by bez tohoto pravidla nešlo z G vygenerovat.

Algorithm 4 CalcObservations(G)

/*gramatika $G = (N, \Sigma, \Pi, S)$ je gramatika bez redundantních pravidel v Chomského normální formě*/
$O = \emptyset$
for all $R \in \Pi$ **do**
  vytvoř slovo w takové, že každý derivační strom tohoto slova používá pravidlo R
  $O = O \cup \{w\}$
end for
return O

CalcDerivations(G) Vytvoří množinu derivačních stromů odpovídajících množině slov z CalcObservations.

Lemma 2 *Nechť $G = (N, \Sigma, \Pi, S)$ je gramatika bez redundantních pravidel v Chomského normální formě. Maximální počet unikátních neterminálů v derivačních stromech v CalcDerivations(G) je dán vzorcem:*

$((\sum_j (2 \times |O_j| - 2)) + 1)$,

kde O_j jsou řetězce vygenerované algoritmem $CalcObservations(G)$.

Algorithm 5 CalcDerivations(G)

/*gramatika $G = (N, \Sigma, \Pi, S)$ je gramatika bez redundantních pravidel v Chomského normální formě*/
$D = \emptyset$
$O = CalcObservations(G)$
for all $w \in O$ **do**
  přidej jednu derivaci tvaru $S \implies {}^{*}w$ do D
end for
return D

Lemma 3 *Nechť $\hat{L}$ je bezkontextový jazyk, gramatika $G = (N, \Sigma, \Pi, S)$ je bezkontextová gramatika v Chomského normální formě popisující jazyk $\hat{L}$, $O = CalcObservations(G)$.*

1. *$|O|$ je konečná*
2. *$O \subseteq \hat{L}$*
3. *Pokud je O předána BruteForceLearner algoritmu, algoritmus sestrojí konečnou množinu hypotetických jazyků $\hat{H}$ takových, že $\hat{L} \in \hat{H}$.*
4. *Po přijmutí O BruteForceLearner algoritmem již algoritmus nebude vytvářet žádné další možné množiny hypotetických jazyků.*

CalcObservationsGivenLang($\hat{L}$) k danému jazyku $\hat{L}$ vytvoří gramatiku G, která ho popisuje a je bez redundantních pravidel. Vydá množinu slov, která by bez nějakého pravidla z gramatiky G nešla vygenerovat. Tento algoritmus jsme nemuseli implementovat, postup je pouze teoretický. Slouží vlastně k vyhnutí se problému s redundantními pravidly v gramatice.

Algorithm 6 CalcObservationsGivenLang($\hat{L}$)

/*$\hat{L}$ je bezkontextový jazyk*/
$G = (N, \Sigma, \Pi, S)$ /*bezkontextová gramatika bez redundantních pravidel taková, že $L(G) = \hat{L}$*/
return CalcObservations(G)

CalcAdditional($H, \hat{L}$) Ke každé gramatice, která je součástí množiny H a nepopisuje jazyk $\hat{L}$, nalezne slovo, které jazyk dané gramatiky z H a

jazyk $\hat{L}$ odlišuje. Tedy slovo, které do jednoho jazyka patří a do druhého ne. Opět se zde setkáváme s postupem, který nelze implementovat, je jen teoretický.

Algorithm 7 CalcAdditional($H, \hat{L}$)

```
/*H je konečná množina hypotetických gramatik*/
/*L̂ je bezkontextový jazyk*/
O = ∅
for all h_i ∈ H do
  if L(h_i) ≠ L̂ then
    if L(h_i) ⊂ L̂ then
      w je řetězec takový, že w ∈ L̂ ∧ w ∉ L(h_i)
      přidám w do množiny O, označený jako řetězec do jazyka patřící
    else
      if L(h_i) ⊄ L̂ then
        w je řetězec takový, že w ∈ L(h_i ∧ w ∉ L̂)
        přidám w do množiny O, označený jako řetězec do jazyka nepatřící
      end if
    end if
  end if
end for
return O
```

Lemma 4 *Nechť $\hat{L}$ je bezkontextový jazyk. BruteForceLearner algoritmu byla předána množina slov označených, zda do jazyku patří či ne. V tuto chvíli BruteForceLearner obsahuje H jako svoji množinu hypotetických gramatik. $\hat{H}$ je konečná množina hypotetických jazyků takových, že $\hat{H}$ jsou jazyky gramatik z H a $\hat{L} \in \hat{H}$. Množina O je výstupem algoritmu CalcAdditional. $O = CalcAdditional(H, \hat{L})$.*

1. *$|O|$ je konečná*
2. *Prvky množiny O jsou označeny jako do jazyka patřící či nepatřící podle jazyka $\hat{L}$.*
3. *Je-li BruteForceLearner algoritmu předána množina O, odstraní všechny nesprávné gramatiky z množiny hypotetických gramatik.*
4. *Po předání O BruteForceLearner algoritmu nebude tento algoritmus vytvářet již žádné další hypotetické gramatiky.*

CalcCharacteristic($\hat{L}$) Výsledkem tohoto algoritmu je množina slov charakterizující cílový jazyk $\hat{L}$. Jsou vygenerována slova charakterizující pravidla gramatiky popisující cílový jazyk a přigenerována slova, která odliší jazyk cílové gramatiky od jazyků ostatních.

Algorithm 8 *$CalcCharacteristic(\hat{L})$*

/*$\hat{L}$ je bezkontextový jazyk*/
$O^+ = CalcObservationsGivenLang(\hat{L})$
$O^- = \{\}$
$H = PruneGrammars(O^+, O^-)$
$C = O^+ \cup CalcAdditional(H, \hat{L})$
return C

Věta 5 *Nechť $\hat{L}$ je bezkontextový jazyk, $O = CalcCharacteristic(\hat{L})$.*

1. *$|O|$ je konečná*
2. *Prvky množiny O jsou označeny jako do jazyka patřící či nepatřící podle jazyka $\hat{L}$.*
3. *Je-li BruteForceLearner algoritmu předána množina O, algoritmus rozezná právě jazyk $\hat{L}$.*
4. *Po předání O BruteForceLearner algoritmu nebude tento algoritmus vytvářet již žádné další hypotetické jazyky.*

Věta 6 *Nechť $\hat{L}$ je bezkontextový jazyk, $O = CalcObservationsGivenLang(\hat{L})$, $C = CalcCharacteristic(\hat{L})$. Po obdržení C bude celkový počet hypotetických gramatik zkonstruovaných algoritmem*

$$2^{1+(((\sum_j (2\times|O_j|-2))+1)^3+((\sum_j (2\times|O_j|-2)+1)|T(O)|))},$$

kde T(O) je množina terminálních symbolů z O.

Lemma 5 *Jestliže BruteForceLearner algoritmus rozezná bezkontextový jazyk, učiní tak v exponenciálním čase vzhledem ke složitosti cílové gramatiky.*

2.4 Trénovací a testovací data

Nyní jsme schopni sestavit množinu označených řetězců do jazyka patřících a nepatřících, z které je možné odvodit cílový jazyk. Známe také velikost prostoru hypotéz, které byly zvažovány nejméně jedním algoritmem gramatické inference. Tato velikost byla použita jako míra obtížnosti úlohy.
Pokud je jazyk popsán deterministickou gramatikou v Chomského normální formě, tato gramatika neobsahuje žádná redundantní pravidla. Každá derivace slova vygenerovaného pomocí daného pravidla musí používat toto pravidlo, protože každé slovo má pouze jednu derivaci. Proto stačí pro vygenerování množiny slov ekvivalentní k výsledku *CalcObservation* pomocí každého pravidla vygenerovat slovo. To je možné i pokud není gramatika v Chomského normální formě. Toto je způsob, jak získat *CalcObservations* v polynomiálním čase vzhledem k počtu pravidel v cílové gramatice.
Podobná technika je použita pro generování *CalcObservations* u nedeterministických gramatik, protože nedeterministické gramatiky byly vytvořeny přidáním nedeterministických pravidel do deterministických gramatik.
Celá množina slov dostatečná pro rozeznání cílového jazyka byla však generována pomocí algoritmu $CalcCharacteristic(\hat{L})$. Tento algoritmus však kvůli použití algoritmu *PruneGrammars* nemůže být vykonán v polynomiálním čase. Kdyby byla úloha dostatečně jednoduchá, aby bylo možné použít tento algoritmus, byla by zároveň příliš jednoduchá, aby byla použita k testování. Místo toho byla použita Věta 7.

Věta 7 *Po obdržení nekonečného proudu unikátních řetězců správně označených, jako patřící nebo nepatřící do cílového bezkontextového jazyka $\hat{L}$, bude po nějakém konečném čase BruteForceLearner algoritmus obsahovat množinu hypotetických gramatik H takových, že pro všechny $h_i \in H : L(h_i) = \hat{L}$. Po tomto okamžiku již BruteForceLearner nebude měnit své hypotézy.*

Aby bylo možné použít předchozí větu, je zapotřebí znát, kolik dalších řetězců má být dogenerováno. V čase kdy vznikala soutěž Omphalos, nebyl dosud znám způsob, jakým to zjistit. Ze zkušenosti se autoři rozhodli pro dogenerování deseti až dvacetinásobku počtu řetězců z $CalcObservations(G)$.

2.4.1 Postup generování dat

1. K dané gramatice je algoritmem *CalcObservations* vygenerována množina charakteristických slov.

2. K zajištění, aby se dal jazyk této gramatiky správně nalézt, jsou vygenerovány další řetězce z jazyka gramatiky a z jeho doplňku, podle Věty

7 v dostatečném počtu tak, aby nahrazovala řetězce generované algoritmem $CalcAdditional(H, \hat{L})$. Díky těmto dodatečným informacím mohou být algoritmem vyloučeny gramatiky z množiny hypotetických gramatik H, které jsou s daty z $CalcObservations$ konzistentní, ale nejsou ekvivalentní s cílovou gramatikou.

3. Data jsou následně rozdělena přibližně na půl do trénovacích a testovacích dat. A to tak aby trénovací data obsahovala množinu charakteristických slov a počet řetězců v ní byl deseti až dvacetinásobkem velikosti množiny charakteristických slov.

2.4.2 Vylepšení generování testovacích dat

V průběhu soutěže Omphalos bylo zjištěno, že některé úlohy byly vyřešeny pomocí regulární aproximace daného jazyka. Generování testovacích dat bylo proto rozšířeno o další metody. Aby nebylo jednoduché odhadnout daný cílový jazyk pouze jeho regulární aproximací, byly přidány řetězce do jazyka nepatřící, ale jazyku blízké. Například ke každému jazyku byla vytvořena právě regulární aproximace a řetězce z doplňku cílového jazyka, které z ní mohly být vygenerovány, byly přidány a označeny jako do jazyka nepatřící. Dále byla vytvořena bezkontextová gramatika, která generovala nadmnožinu jazyka cílové gramatiky. Z této gramatiky opět byla vygenerována a označena slova z doplňku jazyka.

2.5 Generování gramatiky

Aby bylo možné algoritmy otestovat, jak jsou schopné se naučit nějaký bezkontextový jazyk, je nutné nejdříve vytvořit gramatiku, která nějaký takovýto jazyk bude generovat. To je děláno následujícím způsobem.

1. Je sestavena množina terminálních a neterminálních symbolů.

2. Z terminálů a neterminálů jsou vytvořena bezkontextová pravidla náhodným výběrem terminálů a neterminálů.

3. Nalezneme neterminály, které negenerují terminální slovo a přidáme pravidla, kde takovýto neterminál je na levé straně a na pravé straně je terminální slovo.

4. Nalezneme všechny neterminály, které jsou nedosažitelné ze startovacího neterminálu. Přidáme pravidla taková, že startovací neterminál je na levé straně a daný neterminál je obsažen na pravé straně pravidla.

5. Dále je přidáno pravidlo obsahující centrální rekurzi, aby bylo zajištěno, že vygenerovaný jazyk nebude regulární.

Velikosti množin terminálů a neterminálů, maximální délka pravých stran pravidel a počet pravidel jsou zvoleny podle požadavků uživatele (více v uživatelské příručce).

2.6 Ohodnocení algoritmů a srovnání jejich úspěšnosti

K vývoji jakéhokoliv postupu je dobré mít možnost zjistit, nakolik je úspěšný a případně i jak si stojí mezi ostatními algoritmy. V našem testovacím prostředí je pro toto ohodnocení a porovnání připraveno více různých metod.
Jak již bylo řečeno, každému námi testovanému algoritmu jsou předány dva soubory. Soubor s trénovacími daty a soubor s testovacími daty. Úkolem testovaného algoritmu je označit řetězce v testovacím souboru jako patřící buď do jazyka nebo do doplňku jazyka. Následně je námi ohodnocena správnost tohoto označení.

2.6.1 Procentuální úspěšnost

Tato metoda zobrazí, jaké procento úspěšnosti měl daný algoritmus. Výsledkem je číslo $P = right/all * 100$, kde $right$ je počet správně označených řetězců a all je počet všech řetězců v testovacím souboru.
Výpočet je jednoduchý, ale může nám podávat zkreslenou informaci o skutečných dovednostech algoritmů. V následující metodě měření úspěšnosti se blíže podíváme na možný problém této metody.

2.6.2 Ratio

Procentuální úspěšnost může být v některých případech zavádějící. Například v situaci, kdy by jazyk, který se algoritmus učí, obsahoval většinu slov, které lze nad danou abecedou vygenerovat. Algoritmus by za slova do jazyka patřící považoval všechna slova tvořená danou abecedou. Pak by tento algoritmus mohl dosáhnout dobrého hodnocení, ač by se toho moc nenaučil.
Tento problém řeší ohodnocení ratio. Ratio rozděluje úspěšnost na slovech z jazyka a slovech z doplňku jazyka.
Výsledkem je dvojice (l_+, l_-). Tento postup byl popsán v dokumentu [6].

- $l_+ = right_+/all_+$,

- $l_{-} = right_{-}/all_{-}$,

kde $right_{+}$ je počet správně označených slov jako do jazyka patřící, all_{+} je počet všech slov z testu, která do jazyka patří, stejně tak i $right_{-}$ je počet řetězců správně označených patřících do doplňku jazyka a all_{-} je počet všech slov z doplňku jazyka v testu.

2.6.3 F-measure

F-measure je měřítko z oboru vyhledávání informací. Je definováno pomocí přesnosti a úplnosti získaných informací, které byly vyhledávány. Tento postup byl popsán v dokumentu [8]. F-measure F udává vážený harmonický průměr přesnosti a úplnosti vyhledaných informací.

$F = \frac{2*p*r}{p+r}$

- p udává přesnost vyhledaných informací. Je dána vzorcem :
  $p = \frac{svi}{vi}$
- svi je počet správných vrácených informací.
- vi znamená počet všech vrácených informací.
- r udává úplnost odpovědi, tedy poměr mezi počtem vrácených správných informací a počtem všech správných informací:
  $r = \frac{svi}{si}$
- si je počet všech správných informací.

V našem případě jsou jako správně vrácené informace považovány řetězce z testovacích dat, které byly správně testovaným algoritmem označeny jako do jazyka patřící. Počet všech vrácených informací je počet všech řetězců z testovacích dat, které byly daným algoritmem označeny jako do jazyka patřící. Počet všech správných informací je počet všech řetězců z testovacích dat, které do jazyka patří.

Kapitola 3

Závěr

Naším cílem bylo vytvořit nástroj pro testování algoritmů gramatické inference, který by se stal pomůckou pro vědecké pracovníky vyvíjející právě algoritmy gramatické inference. Aplikace byla vytvořena tak, aby byla pomůckou při vývoji, ale i v publikování dosažených výsledků.
Ve vytvořené aplikaci se dají generovat úlohy různých obtížností. Umožňuje porovnávat schopnosti algoritmu na datech se stěžující se obtížností i schopnosti různých algoritmů mezi sebou. Nástroj také umožňuje vyhodnocení úspěšnosti algoritmů několika statistickými metodami. Výsledky jsou po skončení testu vyobrazeny v tabulce a také v podobě grafu. Tabulka i grafy se dají uložit a dále použít.
Seznam algoritmů, které budou testovány, a souborů s testy, na kterých budou schopnosti algoritmů zkoušeny, se dají uložit a znovu obnovit. To přináší výhodu možnosti opakování stejné sady testů, které se může hodit při postupném vylepšování algoritmu.
Gramatiku i testovací data si může uživatel také zadat sám. Struktura souboru s gramatikou i souborů s trénovacími a testovacími daty je jednoduchá. Takto může uživatel vytvářet data tak, aby vyzkoušel na algoritmu, jak se vypořádá s různými úskalími úloh.
Celá práce byla inspirována soutěží Omphalos v gramatické inferenci bezkontextových jazyků. Prostředí tedy vznikalo na základech, které jsou již známé a dalo by se jistě říci i uznávané.
V neposlední řadě byly k aplikaci přidány již fungující algoritmy regulární inference. Tyto algoritmy byly získány na stránkách soutěže v regulární inferenci Abbadingo. Námi byly pouze upraveny tak, aby odpovídaly rozhraní našeho prostředí. Slouží sice k regulární inferenci, ale k vyzkoušení funkčnosti prostředí plně postačují, dokonce je doporučujeme k porovnání s vlastním řešením, neboť dosahují poměrně dobrých výsledků.
Snad každá práce se dá vylepšovat. I u této aplikace je kde s vývojem pokra-

čovat. Daly by se například implementovat postupy generování dat jazyků kontextových či rekurzivně spočetných. I s těmito daty by se sice tato aplikace vypořádala, ale generovat je neumí, v tuto chvíli by si taková data musel uživatel vytvořit sám. Rozumným rozšířením by určitě bylo i přidání dalších statistických metod pro vyhodnocení úspěšnosti či další kriterium a možnosti výpočtu složitosti úlohy.

Kapitola 4

Uživatelská příručka

Přiložený program je GUI aplikací sloužící k testování algoritmů gramatické inference. Je určen pro algoritmy gramatické inference bezkontextových jazyků, ale dají se v něm testovat algoritmy gramatické inference jazyků všech tříd. Umožňuje vytváření souborů s gramatikami, vytváření trénovacích a testovacích dat k těmto gramatikám a testování algoritmů na těchto datech.

Aplikace je rozdělena do čtyř hlavních částí.

- vytváření gramatiky
- generování trénovacích a testovacích dat
- provedení testu jednoho algoritmu na jedné gramatice
- provedení souhrnného testu více algoritmů na více různých gramatikách

Po spuštění aplikace se zobrazí hlavní okno obsahující právě předcházející možnosti. V následujícím textu budou podrobněji popsány.

4.1 Instalace

V této části si řekneme, jakým způsobem se tato aplikace zprovozňuje.
Pro chod aplikace je nutné mít nainstalovány:

- GNU/Linux
- qt4 + devel balíčky
- gnuplot
- make

- gcc

Postup je jednoduchý, stačí v adresáři test_tool zadat make pro kompilaci zdrojových kódů. Pak se již dá aplikace spustit. Doporučuji spouštění z terminálu, neboť jsou zde zobrazovány přídavné informace, podle kterých můžeme zjistit, co právě aplikace dělá.
Pro zprovoznění algoritmů gramatické inference, které jsou k práci přiloženy, je postup podobný. Stačí v konzoli v adresáři s daným algoritmem spustit příkaz make pro kompilaci. A pak se již dá algoritmus používat.

4.2 Vytváření gramatiky

Uživatel má možnost zapsat vlastní gramatiku, upravit gramatiku již existující, či ji nechat vygenerovat aplikací.
Základním oknem této akce je Grammar generation. Zde je možné gramatiku vygenerovat, uložit, otevřít již existující, spočítat obtížnost gramatiky, či otevřít okno pro zapsání gramatiky vlastní.

4.2.1 Generování gramatiky

Pro generování gramatiky je možné zvolit počet neterminálních symbolů, počet terminálních symbolů, počet pravidel a maximální délku jakou pravidla mohou mít. Další možností je zvolení obtížnosti gramatiky, implicitně obsahuje hodnuotu 5. Pokud poté nezadáme některý z předcházejících parametrů, bude dopočten právě z čísla udávajícího obtížnost gramatiky. Jsou-li však zadány všechny tyto údaje, hodnota obtížnosti gramatiky bude ignorována. Po stisknutí tlačítka Create Grammar je z těchto dat vytvořena bezkontextová gramatika. K této gramatice je rovnou spočtena i její obtížnost.
V editačním okně, ve kterém se zobrazí gramatika, je možné dále tuto gramatiku měnit, je ale nutné před uložením spočítat obtížnost této gramatiky.

4.2.2 Vytváření vlastní gramatiky

V okně Grammar generation je možno v menu zvolit akci own grammar. Po tomto výběru je otevřeno okno Create your grammar. Zde jsou volná okénka, nejdříve pro zapsání neterminálních symbolů a terminálních symbolů, které mohou být zadány jako libovolný řetězec, který však nebude obsahovat mezeru a znak svislítko |. Jednotlivé symboly se oddělují mezerami. Následuje pole pro zadání počátečního neterminálu, který je jedním z předtím uvedených neterminálů.
Poslední nutností je zadání pravidel gramatiky. Tato gramatika musí být

zapsána jako bezkontextová gramatika pomocí námi dříve napsaných neterminálních symbolů Γ a terminálních symbolů Σ. Pravidla by měly být ve tvaru: $X\ w$, kde $X \in \Gamma, w \in (\Gamma \cup \Sigma)^+$. Mezi X a w je mezera.
Po stisknutí ok je gramatika zobrazena a spočítána její obtížnost. Je ji možno dále upravovat.

4.3 Generování trénovacích a testovacích dat

Po výběru test generation je otevřeno okno s možností výběru souboru s gramatikou, ze které chceme vygenerovat trénovací a testovací data. Při stisknutí Create files se nás aplikace ještě dotáže na umístění těchto nově vzniklých souborů.
Následně jsou vytvořeny dva soubory jméno.train a jméno.test. Jméno je uživatelem v dialogu vybrané jméno souboru, kterému je odebrána přípona, pokud ji má.

4.3.1 Formát dat

Soubor s trénovacími či testovacími daty si může uživatel vytvořit i sám. Formát je inspirován formátem dat použitým v soutěži Omphalos a Abbadingo. Na prvním řádku souboru je počet slov, které jsou v souboru. Dále se liší podle toho, zda je to soubor trénovací nebo testovací.
Na každém řádku trénovacího souboru je nejdříve značka, zda slovo do jazyka patří či ne. Náležení do jazyka je dáno 1 a náležení do doplňku jazyka je dáno 0. Za značkou následuje délka slova uvedeného na této řádce. Za těmito informacemi již následuje slovo. Jednotlivé údaje a symboly slov jsou odděleny mezerou.
V testovacím souboru není samozřejmě uvedena informace o náležení do jazyka nebo do doplňku jazyka, jinak je formát stejný.

4.4 Provedení jednoho testu

K provedení jednoho testu se dostaneme výběrem One Test z hlavního okna. Jeden test znamená, že bude provedeno spuštění jednoho algoritmu s jedním trénovacím a testovacím souborem. Po skončení testu bude zobrazen výsledek. Tento výsledek je procentuální úspěšností daného algoritmu na zvoleném testu.
Tato volba se hodí například pro opakované testování vyvíjeného algoritmu.

4.5 Provedení velkého testu

Chceme-li srovnat dovednosti více algoritmů nebo provést test na více datech, potom by volbou z hlavního okna měl být full test. Otevře se okno, kde je možné navolit seznam algoritmů, které chceme testovat a seznam souborů, na kterém algoritmy chceme otestovat. Aby bylo možné jednoduše opakovat testy se stejným složením, dají se seznamy uložit a znovu otevřít.

- V menu test je možné zvolit, v jakém tvaru si přeje uživatel zobrazit výsledky testů.
  - Ty mohou být znázorněny v podobě histogramu s procentuální úspěšností algoritmů.
  - V podobě ratio, kde je každý prvek dvojice znázorněn ve vlastním sloupečku histogramu.
  - Další možností je graf, kde x-ová souřadnice je obtížnost testu a y-ová je procentuální úspěšnost algoritmu na testu.
  - Poslední možností je obdoba předchozí volby, kde y-ovou souřadnici tvoří f-measure.
- Výsledky se dají rozdělit do více grafů nebo ponechat v jednom. Volby jsou následující.
  - Úspěšnost algoritmů na testech se může ponechat v jednom grafu.
  - Nebo se zobrazení výsledků dá rozdělit podle testů, při tomto výběru je pro každý testovací soubor vytvořen zvláštní graf, kde jsou výsledky všech algoritmů na tomto testu.
  - Stejně tak se dají výsledky rozdělit i podle algoritmů tak, že pro každý algoritmus je vytvořen graf, kde jsou uvedeny výsledky tohoto algoritmu na všech zadaných testech.

4.6 Algoritmy vhodné pro testování

Algoritmy, které bychom chtěli tímto nástrojem testovat, musí očekávat dva parametry. Prvním parametrem je soubor s trénovacími daty, druhým soubor s daty testovacími. Od algoritmu je očekáván výstup v podobě souboru obsahujícím na každém řádku slovo z testovacích dat označené 0 či 1 podle toho, zda podle algoritmu náleží nebo nenáleží do jazyka cílové gramatiky. Výstupní soubor by tedy měl mít formát stejný jako soubor s trénovacími daty.

Výstupní soubor by měl být vygenerován do adresáře s testovacím souborem a měl by mít i stejný název, až na změnu přípony z test na result.

4.7 Příklad použití

4.7.1 Gramatika

Vezmeme si příklad jednoduché gramatiky.

$G(\{A, B\}, \{a, b\}, A, \Pi)$

$\Pi = \{A \to a, A \to bB, A \to aA, B \to bB, B \to b\}$

Po uložení bude tato gramatika zapsána v souboru v následující podobě:

```
A B
a b
A
A a |b B |a A
B b B |b

a
a a
b b
b b b

748
```

V prvním řádku jsou vypsány všechny neterminální symboly této gramatiky. V druhém jsou terminální symboly. Na třetím řádku je startovací neterminální symbol gramatiky. Na dalších řádcích jsou přepisovací pravidla gramatiky ve tvaru neterminální symbol na levé straně pravidla mezera pravá strana pravidla. Pokud má více pravidel gramatiky stejnou levou stranu pravidla, jsou k nim patřící pravé strany uvedeny všechny v jednom řádku, odděleny znakem |.

Po gramatice následuje prázdný řádek, pod kterým jsou uvedeny charakteristické řetězce této gramatiky. Na každém řádku je umístěno jedno charakteristické slovo.

Další informací uvedenou v souboru je obtížnost této gramatiky. Ta je od charakteristických slov této gramatiky oddělena prázdným řádkem. Neboť se jedná o velké číslo, je uvedeno ve tvaru $\log_2(N)$, kde N je skutečná složitost gramatiky.

Tyto informace samozřejmě uživatel všechny vypisovat nemusí. Stačí aplikaci zadat v souboru jen část s gramatikou, zbytek si program dopočítá.

Ručně si můžeme vytvořit gramatiku i v okně grammar generation. Při uložení do souboru bude gramatika uvedena v tomto tvaru.

4.7.2 Trénovací a testovací data

Gramatiku již máme vytvořenou, teď si k ní v okně test generation vytvoříme trénovací a testovací data. Po výběru souboru s gramatikou, ke které se budou data tvořit, stiskneme na tlačítko pro spuštění generování. Bude po nás vyžádáno ještě umístění vygenerovaných dat. Předpokládá se, že bude uvedeno jen základní jméno souboru. Vygenerovaným souborům bude přidělena přípona test a train, podle toho zda se jedná o soubor s daty testovacími nebo trénovacími.
K naší gramatice byl vygenerován následující soubor s trénovacími daty.

```
70 2
0 6 a b b a b b
1 10 a a b b b b b b b b
0 5 a a a b a
1 5 b b b b b
1 11 a b b b b b b b b b b
1 2 a a
0 3 a a b
0 2 b a
0 6 b b a a b a
1 7 a a a a b b b
1 6 a b b b b b
1 12 a a a a a a b b b b b b
1 3 b b b
⋮
```

Soubor by pokračoval stejným způsobem dále. Řádků s daty by bylo přesně 70. Tato část však k představě podoby souboru stačí. Číslo 2 je počet terminálních symbolů, které by se mohly v datech vyskytnout. Před řetězcem je na každém řádku uvedeno, zda řetězec patří do jazyka cílové gramatiky, to značí číslo 1. Nebo zda řetězec patří do doplňku jazyka cílové gramatiky, to je vyznačeno číslem 0. Následuje délka řetězce a pak již samotný řetězec.
Jen pro úplnost uvedu i část souboru s testovacími daty.

```
66 2
4 b b b b
4 a b a a
```

```
8 a a a a b b b b
12 b b b b b b b b b b b b
12 a a b b b b b b b b b b
6 b a a a b b
8 a a a a a a a a
8 a a a a a b a a
3 b a b
7 a a a a a a a
6 b a b a b b
7 b a b b a b a
7 a a a b b b b
9 a a b b b b b b b
⋮
```

V souboru s testovacími daty není záměrně uvedeno, zda řetězec patří do jazyka či ne.

4.7.3 Testování algoritmů

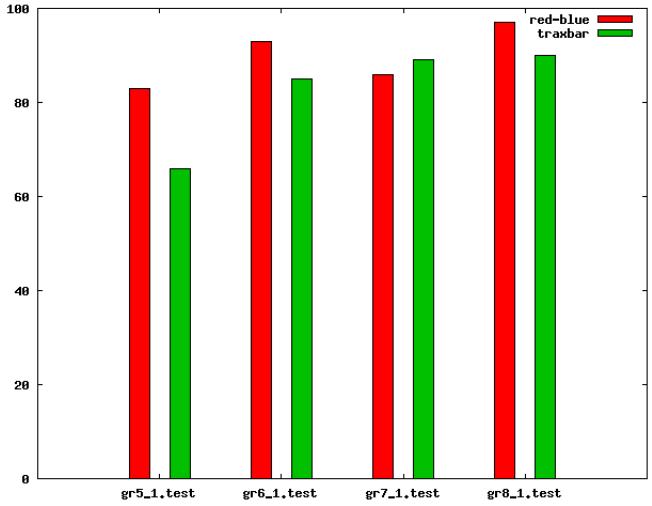
Provedení jednoho testu je poměrně nezajímavé, výsledkem je pouze jedno číslo, které udává procentuální úspěšnost daného algoritmu na daném testu. My se proto nyní budeme zabývat rovnou příkladem provedení testu souhrnného v okně full test.
Naše gramatika byla příliš jednoduchá, proto na ní všechny ozkoušené algoritmy měly stoprocentní úspěšnost. Zajímavější bude porovnání algoritmů na obtížnějších datech.
Příklad 4.1 ukazuje dva algoritmy na třech souborech. Byla zvolena možnost zobrazení histogramem se všemi algoritmy a soubory pohromadě.

Ke grafům je vždy vyobrazena tabulka, jak si který algoritmus vedl na jakém testu.

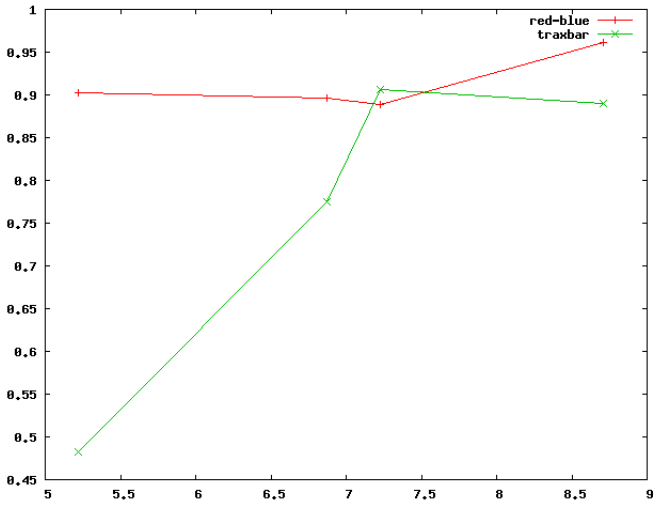
Na obrázku 4.2 je zvoleno vyobrazení f-value. Na souřadnici X je nanášena obtížnost testu, osu Y tvoří právě dosažená hodnota f-value.

Na obrázku 4.3 jsou výsledky již na jiných datech s možností positive x negative, která vyobrazuje hodnoty ratio. Tedy s rozdělenými hodnotami, jak byly algoritmy úspěšné na řetězcích z jazyka gramatiky a jak na řetězcích z doplňku jazyka gramatiky. Je vidět, že vyobrazených hodnot je již hodně, zde tedy můžeme využít rozdělění na jednotlivé algoritmy 4.5 nebo testy 4.4.

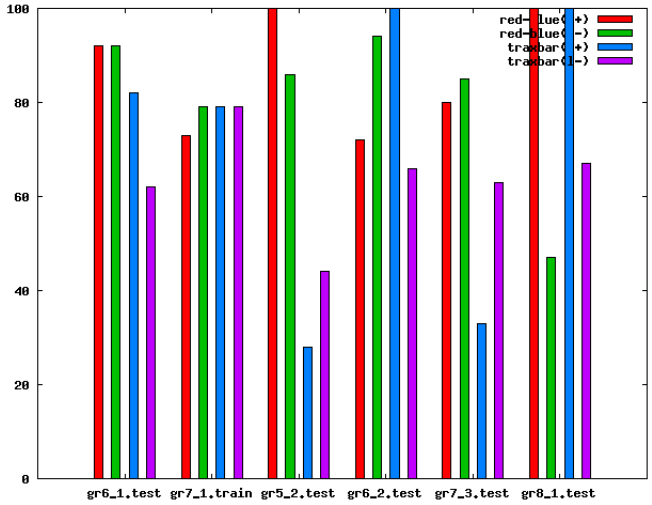
Obrázek 4.1: nastavení histogram, all together



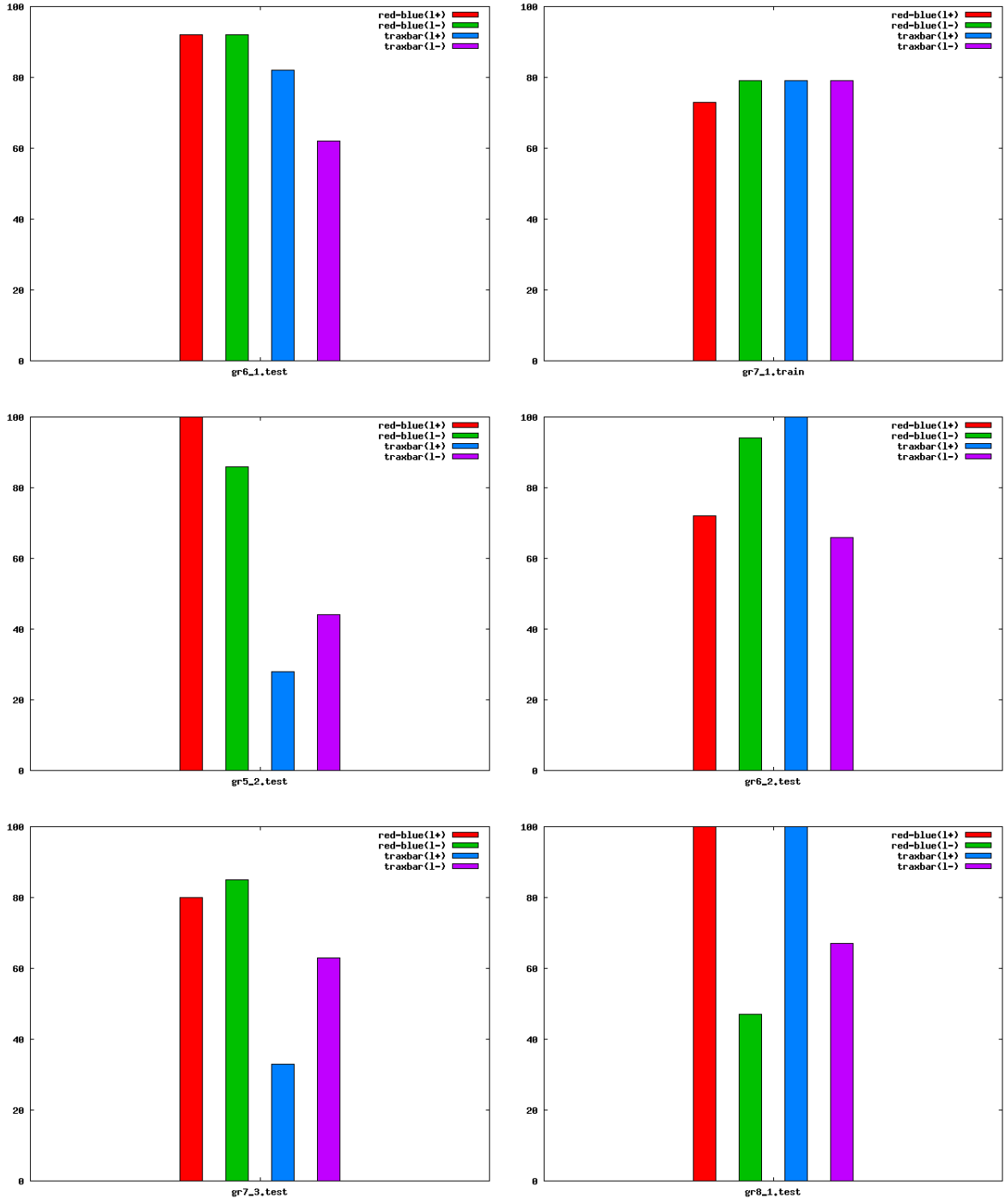
Obrázek 4.2: nastavení f-value, all together



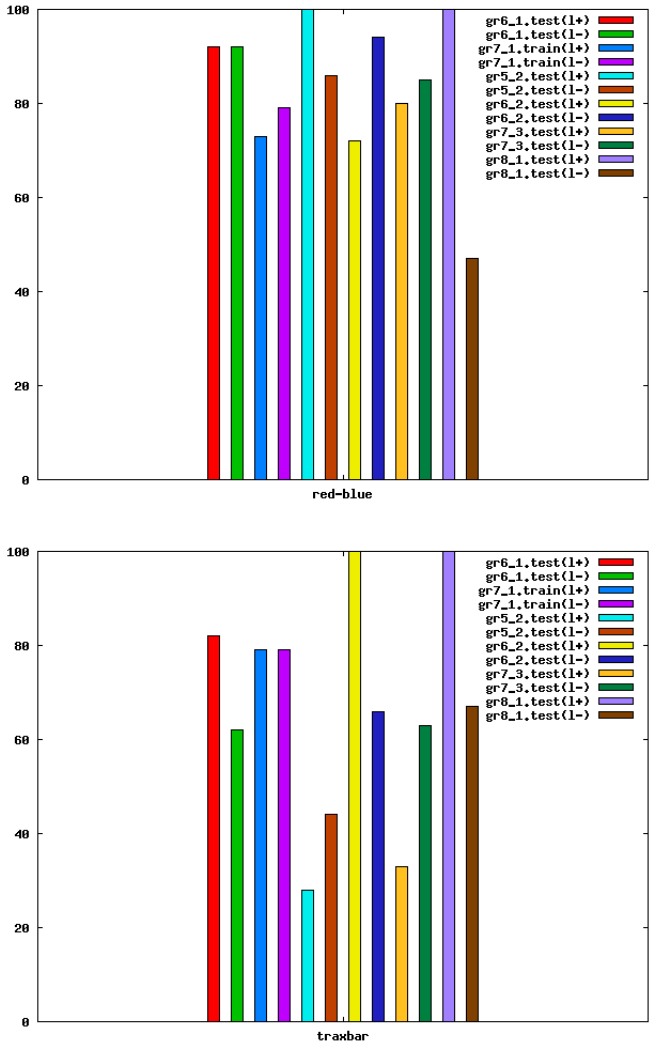
Obrázek 4.3: nastavení ratio, all together



Obrázek 4.4: nastavení ratio, separated by tests



Obrázek 4.5: nastavení ratio, separated by algs



Kapitola 5

Programátorská dokumentace

V této části budeme popisovat naší aplikaci z hlediska programátora.
Základní třídy aplikace byly vyvíjeny v Microsoft Visual Studio 2005, dále bylo použito vývojové prostředí NetBeans 6.0. Aplikace je určena pro systémy na bázi UNIXu. Zkompilována a testována byla v openSUSE 11.0. Aplikace má dvě hlavní části.

- popis tříd a pomocných funkcí
- popis oken a dialogů, která využívají předcházejících tříd

Každá tato část je popsána v samostatné sekci jí věnované. Následně jsou vyjmenovány prostředky, které aplikace vyžaduje. V poslední části této kapitoly je zmíněna možnost použití předpřipravených algoritmů pro testování.

5.1 Popis tříd a pomocných funkcí

Třídou popisující bezkontextovou gramatiku je třída bkg implementovaná v souborech `bkg.h` a `bkg.cpp`. Tato třída obsahuje třídy reprezentující pravidla, převod a množinu terminálních a neterminálních symbolů.
Třídou pro uchování pravidel bezkontextové gramatiky je třída pravidla, která je popsána v souborech `pravidla.h` a `pravidla.cpp`.
Pro zmíněný převod uvnitř bkg slouží třída ObousmSl. Má funkci obousměrného slovníku. Aby při práci s pravidly nemuselo být pracováno s řetězci, jsou řetězce reprezentující terminální a neterminální symboly zavedeny do slovníku a přeloženy na typ symbol, což je v našem případě int. S původními řetězci se opět setkáme pouze při výstupu pro uživatele, interně se stále pracuje s typem symbol. Třída pro převod je v souboru `prevod.h`.
Funkce pro náhodné vygenerování bezkontextové gramatiky je popsána v

souborech `GenerovaniGramatiky.h` a `GenerovaniGramatiky.cpp`. Gramatiky generované pomocí funkcí uvnitř tohoto souboru jsou vytvářeny způsobem popsaným v části generování gramatiky. Jak je z popisu vidět, jsou již v redukovaném tvaru. Gramatiky, které získáme od uživatele, do redukovaného tvaru musíme převést, k tomu slouží funkce redukce, která je deklarována v souboru `redukce.h` a implementována v `redukce.cpp`.
Po získání gramatiky je třeba spočítat její obtížnost. K tomu slouží fuknce SlozitostGramatiky deklarovaná v `komplexita.h` a implementována v `komplexita.cpp`.
Pro vygenerování trénovacích a testovacích dat slouží funkce uvnitř souboru `TestovaciData.h`.
Testování gramatik probíhá pomocí funkcí uvnitř souboru `test.h`. Při testování je potřeba vyhodnotit, zda dané slovo patří do jazyka gramatiky či ne. To se provádí pomocí algoritmu CYK, tento algoritmus je zapsán uvnitř souboru `CYK.h`. Algoritmus CYK vyžaduje, aby byla bezkontextová gramatika v Chomského normální formě. Převod bezkontextové gramatiky do Chomského normální formy dělá funkce CNF deklarovaná v souboru `CNF.h` a definovaná uvnitř `CNF.cpp`.

5.2 Okna a dialogy

GUI celé aplikace bylo vytvořeno pomocí Qt Designeru 4.4.0 Open Source Edition.
V souboru `main.cpp` je pouze funkce main, která vytváří a zobrazuje MainWindow, hlavní okno aplikace. Toto okno slouží pouze k výběru následující činnosti. Je popsáno v souboru `MainWindow.h` a `MainWindow.cpp`.
Z hlavního okna je možné se dostat na generování gramatiky, generování testů k existující gramatice, provádění jednoho malého testu a k provádění souhrnného testu.
Generování gramatiky je vykonáváno v okně GenG. To je popsáno v `GenG.cpp` a `GenG.h`, zde jsou používány třídy a funkce deklarované v `bkg.h`, `GenerovaniGramatiky.h`, `redukce.h` a `komplexita.h`. Z okna pro generování gramatiky je otevíráno okno pro zapsání vlastní gramatiky TypeGrammar, popsané v `TypeGrammar.h`. To opět používá třídy funkce z `bkg.h`.
Generování testů k vybraným gramatikám je prováděno v okně TrainTest, které je deklarováno v souboru `TrainTest.h`. Opět používá třídy a funkce ze souborů `bkg.h`, dále z `TestovaciData.h` a `CNF.h`.
Malý test jednoho algoritmu na jednom souboru je prováděn v okně OneTest, deklarovaném v `OneTest.h`. Toto okno používá třídy a funkce ze souboru `test.h`.

Test velký s možností otestování více algoritmů na více testovacích souborech je vykonáván z okna FullTest, které je deklarované v `FullTest.h`. Toto okno využívá tříd a funkcí z `bkg.h` a `test.h`. Výsledky testu jsou po jeho skončení zobrazeny v okně imageViewer popsaném v souboru `imageViewer.h`.

5.3 Algoritmy k testování

K řešení byly přidány i algoritmy, na kterých se testování dá vyzkoušet, případně s nimi uživatel může porovnat své řešení. Tyto algoritmy jsou algoritmy regulární inference získané ze stránek soutěže Abbadingo v regulární inferenci. Byly pouze upraveny tak, aby jejich spouštění odpovídalo našim požadavkům.
I když jsou přidané algoritmy určeny pro regulární jazyky, dosahují dobrých výsledků i na jazycích bezkontextových. Jsou tedy pro uživatele aplikace vhodné i k porovnání jejich schopností se schopnostmi jejich vlastního řešení.

5.4 Požadavky

K úplnosti musím dodat, že k vykreslování grafů aplikace používá gnuplot, který musí být na počítači nainstalován. Dále vyžaduje knihovny k běhu qt4.

Literatura

[1] Starkie B., Coste F., Van Zaanen M.: *Progressing the State-of-the-art in Grammatical Inference by Competition*, AI Communications, 2005, 93-115.

[2] Starkie B., Coste F., Van Zaanen M.: *Background theory* [online], citováno 24.5.2009, url: http://www.irisa.fr/Omphalos/generation2.html.

[3] *Omphalos Context-Free Language Learning Competition* [online] citováno 24.5.2009, url: http://www.irisa.fr/Omphalos/.

[4] Hopcroft J. E., Motwani R., Ullman J. D.: *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 2001.

[5] Barták R.: *Automaty a gramatiky*, 2007, [online] citováno 24.5.2009 http://kti.mff.cuni.cz/~bartak/automaty/prednaska.html.

[6] Hoffmann P.: *Učenie reštartovacích automatov genetickými algoritmami*, Diplomová práce MFF UK, Praha, 2003.

[7] *Abbadingo One: DFA Learning Competition* [online], citováno 12.10.2008, url: http://abbadingo.cs.unm.edu/dfa.html.

[8] *Information retrieval*, [online] citováno 24.5.2009, url: http://en.wikipedia.org/wiki/Information_retrieval.