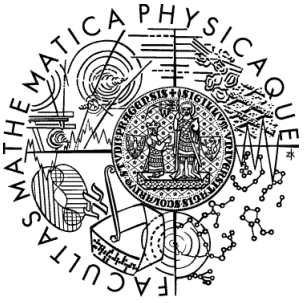
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jan Vyšohlíd

Prostředí pro testování algoritmů pro učení automatů

Kabinet software a výuky informatiky

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

2008

Děkuji především svému vedoucímu RNDr. Petru Hoffmannovi za ochotu a trpělivost, s jakou mi odpovídal na mé dotazy, za připomínky, rady a náměty vedoucí ke zlepšení kvality práce, za čas strávený konzultacemi se mnou a celkovou pomoc při psaní této práce. Dále děkuji všem citovaným autorům za jejich díla, z nichž jsem mohl čerpat. Bez všech těchto lidí by práce nedostala svou nynější podobu.

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

V Praze dne 28.7.2008 Jan Vyšohlíd

Obsah

Název práce: Prostředí pro testování algoritmů pro učení automatů
Autor: Jan Vyšohlíd
Katedra (ústav): Kabinet software a výuky informatiky
Vedoucí bakalářské práce: RNDr. Petr Hoffmann
E-mail vedoucího: petr.hoffmann@mff.cuni.cz

Abstrakt: V předložené práci studujeme metody pro testování algoritmů regulární inference. Nejprve jsou uvedeny některé teoretické poznatky z oblasti konečných automatů a regulární inference. Dále jsou představeny některé algoritmy pro učení konečných automatů, jejich principy a použité testovací metody, jež zjišťují kvalitu algoritmů testováním výsledných automatů. V dalším textu je pak vysvětlen způsob generování trénovacích a testovacích dat, popsán formát pro uložení těchto dat a pro uložení konečných automatů a nakonec také samotný průběh testování algoritmů. Součástí práce je rovněž aplikace, která uvedené testovací metody implementuje a výsledné statistiky ukládá ve zvoleném formátu. V dodatcích přikládáme uživatelskou a programátorskou dokumentaci k této aplikaci.

Klíčová slova: konečný automat, regulární inference, učící algoritmus, testovací metoda, aplikace

Title: An application for testing automata learning algorithms
Author: Jan Vyšohlíd
Department: Department of Software and Computer Science Education
Supervisor: RNDr. Petr Hoffmann
Supervisor's e-mail address: petr.hoffmann@mff.cuni.cz

Abstract: In the present work we study methods for testing regular inference algorithms. First there are introduced some theoretical basics for finite state automata and regular inference. Next we present some finite state automata learning algorithms, their principles and used testing methods, which find out algorithms quality via testing resulting automata. Following text makes the training and testing data generating process clear, describes format for saving this data and for saving finite state automata and finally the algorithms testing run alone, too. The application implementing present testing methods is a part of this work as well. It saves result statistics in chosen format. In appendices we append user and programmer documentation for this application.

Keywords: finite state automaton, regular inference, learning algorithm, testing method, application

Kapitola 1

Úvod

1.1 Charakteristika práce

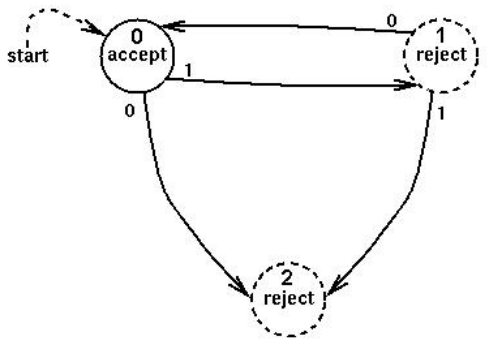
Cílem práce je studovat a navrhnout metody pro testování algoritmů pro učení konečných automatů a navržené postupy implementovat. Problém učení konečných automatů není jednoduchý a existují pro něj různé způsoby zadání i postupy řešení. Tato práce se bude zabývat učením z trénovacích dat. Algoritmy pro učení konečných automatů z trénovacích dat se také nazývají algoritmy regulární inference. Problém regulární inference je velmi zajímavý a užitečný. Má nemalý teoretický význam, ale také široké spektrum aplikací jako je např. identifikace sekvenčních procesů, rozpoznávání vzorů, zpracování řeči a přirozeného jazyka, exploratorní sekvenční analýza, umělá inteligence nebo data mining. Máme dána nějaká data z jazyka, který neznáme. Z nich chceme získat automat, který pozitivní příklady přijímá a negativní příklady zamítá. Jinými slovy hledáme automat konzistentní s těmito daty. Řešení tohoto problému ale není jednoznačné, jak je vidět na následujícím příkladu.

Příklad dvou automatů konzistentních se stejnými trénovacími daty, které můžeme oba považovat za řešení problému regulární inference, rozpoznávajících různé jazyky:

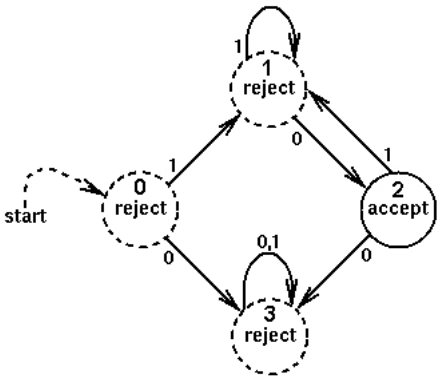
Trénovací data:

pozitivní příklady: {10, 101010}
negativní příklady: {100}

Automat 1:



Automat 2:



V ideálním případě by výsledný automat měl rozpoznávat jazyk, z něhož byla trénovací data nagenerována. Automat může ale přijímat např. nadmnožinu tohoto jazyka a přesto bude odpovídat trénovacím datům. To znamená, že existuje velké množství automatů, které rozpoznávají slova z tohoto jazyka a neví se tedy, který konkrétní automat je nejlepší a měl by být vybrán jako výsledný. Možným přístupem k řešení tohoto problému je Occamova břitva [9]. Podle ní je řešení tím lepší, čím je jednodušší. V našem případě např. automat mající méně stavů můžeme považovat za lepší než automat s více stavy. Automat s minimálním počtem stavů mezi všemi automaty rozpoznávajícími daný jazyk je označován jako mDFA a bude definován v příští kapitole. Ovšem jak je uvedeno v [8], problém hledání mDFA konzistentního s trénovacími daty je NP-úplný, proto se v praxi dle [5] hledá tzv. optimální automat, který se mDFA snaží nějakým způsobem přiblížit. Protože algoritmů regulární inference bylo vymyšleno poměrně hodně a vznikají stále nové, bylo by užitečné porovnat existující algoritmy i algoritmy nově navržené a právě to by měla umožnit přiložená aplikace (dále jen aplikace).

Tato aplikace implementuje zmíněné testování kvality algoritmů regulární inference pomocí vybraných statistických metod, které se zaměřují na různá hlediska přiblížení uvedenému cílovému automatu, podle něhož byla generována trénovací a testovací data. Testovaný algoritmus je dodán v podobě externího programu, kterému aplikace předá nagenerovaná trénovací data v pevném formátu a obdrží od něj (deterministický) konečný automat v pevném formátu. Tyto formáty budeme dále nazývat Abbadingo formát dat a Abbadingo formát automatu, protože byly použity v [1] a odtud jsou také převzaty. Automat pak aplikace otestuje zvolenou statistickou metodou a vypočítá statistické údaje, které je možné uložit do souboru ve formátu vhodném pro vložení do publikací (např. obrázek či LaTeXový kód). Součástí aplikace je také několik programů s algoritmy regulární inference stažených z [2], aby bylo možné porovnat nové algoritmy regulární inference se schopnými a vyzkoušenými. Protože tyto algoritmy byly k dispozici pouze pro UNIXové systémy, jsou v rámci práce upraveny také pro MS Windows.

Práce je určena především vědeckým pracovníkům vyvíjejícím algoritmy regulární inference, kteří potřebují otestovat své postupy různými statistickými metodami nebo je prezentovat v publikacích. Dále jistě bude užitečná těm, kteří využívají algoritmy regulární inference, aby si pro své účely mohli vybrat vhodný algoritmus na základě srovnání pomocí přiložené aplikace. Byla ale napsána pro všechny, kteří

se o zpracovanou problematiku zajímají nebo si chtějí jen zkusit otestovat kvalitu některého z algoritmů. Jak je popsáno v kapitole 4, aplikaci lze snadno rozšířit např. o další testovací metody, způsoby generování dat nebo ji upravit pro vlastní specifické účely použití.

1.2 Stručný popis kapitol

Kapitola 2 seznamuje se základními definicemi a větami z oblasti konečných automatů a regulární inference a s principy algoritmů regulární inference. V kapitole 3 jsou popsány zvolené statistické metody pro testování jejich kvality, Abbadingo formáty trénovacích a testovacích dat a automatu a nakonec samotný průběh testování. Kapitola 4 shrnuje a hodnotí získané poznatky. Dodatky A a B obsahují uživatelskou a programátorskou dokumentaci k nástroji, který implementuje popsané testovací metody. Na závěr práce je přiložen seznam použité literatury.

Kapitola 2

Základní teorie

Tato kapitola seznamuje se základními definicemi a větami z oblasti konečných automatů a regulární inference a s principy algoritmů regulární inference.

2.1 Konečné automaty

Tato podkapitola podává základní definice a věty z oblasti konečných automatů. Níže definujeme řetězce [4], prefix řetězce [6] a jazyk [4].

Definice 2.1.1 *Nechť Σ je libovolná konečná množina (nazýváme ji abeceda), pak Σ^+ označuje množinu všech konečných neprázdných posloupností utvořených z prvků množiny Σ, λ označuje prázdnou posloupnost a definujeme $\Sigma^* = \Sigma^+ \cup \{\lambda\}$. Prvky Σ^* se nazývají řetězce.*

Definice 2.1.2 *Nechť Σ je konečná abeceda. Nechť u, v jsou prvky z množiny Σ^* (tedy řetězce nad abecedou Σ). Řekneme, že u je prefixem v, pokud existuje $w \in \Sigma^*$ tak, že $uw = v$.*

Definice 2.1.3 *Je-li Σ konečná abeceda a $L \subseteq \Sigma^*$, pak L nazýváme jazykem nad abecedou Σ.*

K popisu jazyků používáme automaty. Ty můžeme uspořádat podle jejich vyjadřovací síly. Toto uspořádání se nazývá Chomského hierarchie. Konečné automaty jsou v tomto uspořádání nejníže a mají tedy nejmenší vyjadřovací sílu. V následujícím textu definujeme konečný automat, jazyk rozpoznávaný konečným automatem [4] a prefix tree acceptor [10].

Definice 2.1.4 *Konečným automatem nazýváme každou pětici $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, kde Q je konečná neprázdná množina stavů (stavový prostor), Σ je konečná neprázdná množina vstupních symbolů (vstupní abeceda), δ: $Q \times \Sigma \to Q$ je přechodová funkce, $q_0 \in Q$ je počáteční stav a $F \subseteq Q$ je množina koncových stavů.*

Definice 2.1.5 *Pro konečný automat $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ rozšíříme přechodovou funkci δ: $Q \times \Sigma \to Q$ na zobecněnou přechodovou funkci δ^*: $Q \times \Sigma^* \to Q$ takto:*

1. $\delta^*(q, \lambda) = q$ *pro každé* $q \in Q$,

2. $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$ *pro každé* $q \in Q, w \in \Sigma^*, a \in \Sigma$.

Jazykem rozpoznávaným konečným automatem $\mathcal{A}$ *pak nazveme jazyk*

$$\mathrm{L}(\mathcal{A}) = \{w \mid w \in \Sigma^* \ \& \ \delta^*(q_0, w) \in F\}.$$

Říkáme, že slovo $w \in \Sigma^*$ *je přijímáno automatem* $\mathcal{A}$*, právě když* $w \in \mathrm{L}(\mathcal{A})$.

Definice 2.1.6 *Prefix tree acceptor vzhledem k* $I_+(\subset \Sigma^*)$ *neprázdné množině řetězců je konečný automat* $\mathrm{PTA}(I_+) = (Q, \Sigma, \delta, q_0, F)$*, kde* $Q = \mathrm{Pr}(I_+)$ *je množina všech prefixů slov z* $I_+, q_0 = \lambda$ *a* $\delta(u, a) = ua$*, přičemž* $u, ua \in Q, u \in \Sigma^*, a \in \Sigma$.

Prefix tree acceptor můžeme vnímat jako strom, v němž každá cesta z kořene do přijímacího stavu odpovídá řetězci z dané množiny I_+. Dále definujeme mDFA [5] a jazyk rozpoznatelný konečným automatem [4].

Definice 2.1.7 *Nechť* $\mathrm{mDFA}(L)$ *označuje počtem stavů minimální deterministický konečný automat přijímající jazyk* L.

Definice 2.1.8 *Řekneme, že jazyk* L *je rozpoznatelný konečným automatem, jestliže existuje konečný automat* $\mathcal{A}$ *takový, že* $L = \mathrm{L}(\mathcal{A})$.

Existují různé způsoby, kterými můžeme jazyky rozpoznávané konečnými automaty charakterizovat. Jedním z nich jsou regulární jazyky [4].

Definice 2.1.9 *Třída* $\mathrm{RJ}(\Sigma)$ *regulárních jazyků nad abecedou* Σ *je nejmenší třída jazyků nad abecedou* Σ *splňující tyto podmínky:*

1. $\emptyset \in \mathrm{RJ}(\Sigma)$ *a* $\{a\} \in \mathrm{RJ}(\Sigma)\ \forall a \in \Sigma$.

2. $L_1, L_2 \in \mathrm{RJ}(\Sigma) \Rightarrow L_1 \cup L_2 \in \mathrm{RJ}(\Sigma)$*, kde operace* $\cup$ *je sjednocení jazyků.*

3. $L_1, L_2 \in \mathrm{RJ}(\Sigma) \Rightarrow L_1 \,.\, L_2 \in \mathrm{RJ}(\Sigma)$*, kde operace* $.$ *je součin (zřetězení) jazyků.*

4. $L \in \mathrm{RJ}(\Sigma) \Rightarrow L^* \in \mathrm{RJ}(\Sigma)$*, kde operace* * *je iterace jazyka.*

Vztah mezi regulárními jazyky a konečnými automaty je dán větou níže [4].

Věta 2.1.10 (Kleene) *Libovolný jazyk je regulární, právě když je rozpoznatelný konečným automatem.*

Dalším způsobem charakterizace jazyků rozpoznávaných konečnými automaty jsou regulární výrazy [4].

Definice 2.1.11 *Množinu* $\mathrm{RV}(\Sigma)$ *regulárních výrazů nad abecedou* $\Sigma = \{a_1, ..., a_n\}$ *definujeme jako nejmenší množinu slov v abecedě* $\{a_1, ..., a_n, \emptyset, \lambda, +, ., *, (,)\}$ *splňující tyto podmínky [kde* $\emptyset, \lambda, +, ., *, (,)$ *jsou symboly nepatřící do* Σ*]:*

1. $\emptyset \in \mathrm{RV}(\Sigma)$,
   $\lambda \in \mathrm{RV}(\Sigma)$,
   $a \in \mathrm{RV}(\Sigma)$ *pro každé* $a \in \Sigma$.

2. $\alpha, \beta \in \mathrm{RV}(\Sigma) \Rightarrow (\alpha + \beta) \in \mathrm{RV}(\Sigma), (\alpha \ . \ \beta) \in \mathrm{RV}(\Sigma), (\alpha)^* \in \mathrm{RV}(\Sigma)$.

Vztah mezi regulárními jazyky a regulárními výrazy vyjadřuje definice níže [4].

Definice 2.1.12 *Každý z regulárních výrazů označuje jistý regulární jazyk. Výraz* $\emptyset$ *označuje prázdný jazyk, výraz* λ *označuje jazyk* $\{\lambda\}$*, pro* $a \in \Sigma$ *označuje výraz* a *jazyk* $\{a\}$*. Jestliže* α, β *jsou výrazy označující po řadě jazyky* L_1, L_2*, potom* $(\alpha + \beta)$ *označuje* $L_1 \cup L_2$*,* $(\alpha \ . \ \beta)$ *označuje* $L_1 \ . \ L_2$ *a* $(\alpha)^*$ *označuje* $(L_1)^*$*.*

2.2 Regulární inference

Tato podkapitola obsahuje základní definice z oblasti regulární inference a seznamuje s principy algoritmů regulární inference. Nejprve je potřeba vědět, jak vypadají trénovací data, z nichž bude učení automatů probíhat a co je vlastně učení a regulární inference [6].

Definice 2.2.1 *Trénovacími daty pro jazyk* L *nazveme* $I = I_+ \cup I_-$ *množinu příkladů sestávající z pozitivního vzorku* I_+*, tzn. konečné podmnožiny z jazyka* L*, a negativního vzorku* I_-*, tzn. konečné množiny slov nenáležejících do jazyka* L *(neboli konečné podmnožiny doplňku jazyka* L*). Množina* I_- *může být v některých případech prázdná.*

Definice 2.2.2 *Nechť* I_+ *a* I_- *jsou disjunktní podmnožiny množiny* Σ^**, které označují pozitivní a negativní učící vzorky inferenčního algoritmu a* $I = I_+ \cup I_-$*. Učením nazýváme proces hledání konečného automatu* $\mathcal{A}$ *takového, že vzorky z množiny* I_+ *jsou přijímány automatem* $\mathcal{A}$ *a vzorky z množiny* I_- *jsou automatem* $\mathcal{A}$ *zamítány. Regulární inferencí nazýváme proces učení z trénovacích dat* I*.*

Některé algoritmy regulární inference požadují, aby množina I_+ byla tzv. strukturálně kompletní [6].

Definice 2.2.3 *Množina pozitivních vzorků* I_+ *je nazývána strukturálně kompletní vzhledem ke konečnému automatu* $\mathcal{A}$*, pokud existují přijímající výpočty pro slova z množiny* I_+ *takové, že platí:*

1. *Každý přechod* $\mathcal{A}$ *je použit nejméně jednou při přijímání řetězců z* I_+*.*

2. *Každý prvek* F *(koncový stav) je přijímací alespoň pro jeden řetězec z* I_+*.*

Pro problém regulární inference definované výše existuje hodně řešení. My z nich chceme vybrat pouze jedno, které by nám nejvíce vyhovovalo. Jednou z možností, která byla již okomentována v úvodu práce, je hledat co nejmenší výsledný automat co do počtu stavů. Následuje formalizace této možnosti řešení [5].

Definice 2.2.4 *Mějme množiny pozitivních a negativních vzorků I_+ a I_-. Potom konečný automat $\mathcal{A}$ je řešením problému regulární inference, pokud jsou splněny následující podmínky:*

1. *I_+ je strukturálně kompletní vzhledem k $\mathcal{A}$.*
2. *$I_- \subseteq \Sigma^* - \mathrm{L}(\mathcal{A})$.*
3. *$\mathcal{A}$ je automat s nejmenším počtem stavů, který splňuje předchozí dvě podmínky.*

Řešení můžeme omezit na mDFA definovaný v předchozí podkapitole konzistentní s I_+ *a* I_-. Ovšem jak je uvedeno v [8], problém hledání mDFA konzistentního s trénovacími daty je NP-úplný, proto se v praxi dle [5] hledá tzv. optimální automat, který se mDFA snaží nějakým způsobem přiblížit. Právě testováním tohoto optimálního automatu zjišťujeme kvalitu učícího algoritmu podle různých kritérií volbou vhodné testovací metody. Tato kritéria jsou popsána spolu s testovacími metodami v další kapitole.

Další text popisuje dělení a principy algoritmů regulární inference s jejich konkrétními příklady a je zpracován podle [9].

Algoritmy regulární inference můžeme rozdělit do několika skupin. První velkou skupinou jsou algoritmy, které začínají s PTA pro zadané I_+ a slučují stavy za účelem získání výstupního automatu (state merging method [13]). Mohou pracovat s pozitivními i negativními vzorky, např. RPNI (regular positive and negative inference [12]), Traxbar [13] a EDSM (evidence driven state merging [11]), nebo pouze s pozitivními vzorky, např. Alergia [3]. Pokud nejsou k dispozici negativní vzorky, algoritmus musí umět rozpoznat jiným způsobem, kdy ukončit slučování stavů (u algoritmu Alergia je to test podobnosti funkce stavů, další možností je použít Occamovu břitvu [9]), v opačném případě k tomu používá právě negativní vzorky.

Do druhé velké skupiny patří genetické algoritmy nebo jiné algoritmy založené na evoluci, např. GARI (genetic algorithm for regular inference [9]).

Další algoritmy jsou většinou modifikací principů zmíněných skupin nebo je vylepšují přidanými prvky, např. různými heuristikami. Jednou z nich je např. minimal message length (MML [9]).

Kapitola 3

Testování algoritmů

V této kapitole jsou popsány zvolené statistické metody pro testování jejich kvality, Abbadingo formáty trénovacích a testovacích dat a automatu a nakonec samotný průběh testování.

3.1 Testovací metody

V této podkapitole jsou uvedeny testovací metody, které prověřují kvality algoritmů regulární inference podle různých hledisek, použité v aplikaci.

Všechny metody mají několik společných rysů, které je vhodné úvodem popsat. Do výstupního souboru se statistikami je do tabulky vypisována průměrná respektive minimální/průměrná/maximální procentuální úspěšnost testování pro danou metodu a zadaný počet pokusů nebo je jako výstup vykreslen graf. Tyto výstupy lze také vzájemně kombinovat. Díky opakování testování na různých datech je z tabulky nebo grafu přesněji vidět kvalita automatu od programu s algoritmem, než kdyby testování proběhlo pouze jednou. Genetické (evoluční) algoritmy jsou obecně nedeterministické, proto se při testování použije vždy průměr z 10 běhů algoritmu. Tento počet běhů byl pro testování zvolen podle [5]. V následujícím seznamu budou jednotlivé metody popsány spolu s vysvětlením jejich využití:

1. Metoda **ověření konzistence automatu s trénovacími daty**. Používá k tomu stejná trénovací data, jaká byla předána programu s algoritmem. I když se toto testování může na první pohled zdát nesmyslné, protože by např. stačilo, aby automat přijímal jazyk sestávající právě z pozitivních vzorků, poslouží hlavně v prvních fázích vývoje programu, kdy si autor není jist jeho správností. Může se tedy stát, že program dostane trénovací data a na výstup vydá automat, který s nimi není konzistentní.

2. Metoda **testování schopnosti automatu klasifikovat testovací data**. Ta jsou disjunktní s trénovacími daty a lze tedy pomocí nich odhadnout, jak dobře automat zobecňuje na nová data, která předtím ještě neviděl, protože nebyla obsažena v trénovacích datech.

3. Metoda **Abbadingo** podobná předchozí metodě je převzata z [11]. Předchozí metoda z ní dokonce vychází a je její obecnější variantou. Ačkoli je metoda Abbadingo speciálním případem předchozí metody, je přesto velmi užitečná a to proto, že umožňuje jednoduše provést test přesně podle soutěže Abbadingo [11]. Tato metoda také testuje schopnost automatu klasifikovat testovací data jako předchozí metoda, ale dělá to pouze pro automaty s počty stavů 64, 128, 256 a 512, které pracují s binárními vzorky. V aplikaci je navíc upravena tak, že může generovat 1 až 4 z těchto automatů. V článku [11] se považuje problém za vyřešený (automat od programu s algoritmem je dostatečně kvalitní), pokud je automatem správně klasifikováno více než 99% testovacích vzorků. V aplikaci dostáváme místo toho výstup, který je popsán v úvodu této podkapitoly. Použité počty trénovacích dat v metodě Abbadingo pro všechny kombinace počtu stavů a složitosti jsou uvedeny v Tabulce 1, která je převzata podle Tabulky 1 z článku [11]. Jak přesně metoda pracuje je popsáno v uživatelské dokumentaci v dodatku a ukázáno na příkladě.

		složitost trénovacích dat			
		IV	**III**	**II**	**I**
	64	4456	3478	2499	1521
velikost	128	13894	10723	7553	4382
automatu	256	36992	28413	19834	11255
	512	115000	87500	60000	32500

Tabulka 1. Počty trénovacích dat v Abbadingu.

4. Metoda **porovnání počtu stavů zadaného nebo vygenerovaného automatu s počtem stavů automatu od programu s algoritmem**. Právě tato metoda testuje přiblížení mDFA z hlediska počtu stavů. Procentuální úspěšnost této metody ale může být i větší než 100%, protože jednak automat, z něhož byla generována trénovací data nemusel být mDFA pro daný jazyk a navíc zůstává při testování touto metodou otázkou, zda automat od programu přijímá stejný jazyk jako zadaný či vygenerovaný automat. Je tedy dobré tuto metodu kombinovat s metodou testování schopnosti automatu klasifikovat testovací data.

5. Metoda **porovnání MML (minimal message length) pro automat z aplikace a automat od algoritmu regulární inference**. MML je spočítána podle vzorce

$$M\log_2(N)-\log_2((N-1)!)+\sum_{j=1}^{N}(\log_2((t_j+V-1)!)-\log_2((V-1)!)-\sum_{i=1}^{V}\log_2(n_{ij}!))$$

uvedeného v [9], kde M je počet všech přechodů, N počet stavů, V velikost abecedy zvýšená o 1, t_j počet odchodů ze stavu j při procházení trénovacích dat a n_{ij} počet odchodů ze stavu j symbolem i při procházení trénovacích dat. Vypočítané MML jsou porovnány a určí se z nich kvalita automatu od

programu s algoritmem. Tato metoda je po předchozí metodě dalším způsobem měření jednoduchosti výsledného automatu. MML zjednodušeně řečeno určuje, jak dlouhá zpráva by byla potřeba, aby byla co nejkratší a zároveň dokázala příjemci popsat automat i trénovací data zakódovaná pomocí automatu tak, aby je podle ní uměl zrekonstruovat.

3.2 Abbadingo formáty

V této podkapitole jsou popsány Abbadingo formáty trénovacích a testovacích dat a automatu i s konkrétními příklady podle [1].

Abbadingo formát trénovacích a testovacích dat má v hlavičce uveden počet řetězců, z nichž se data skládají a počet symbolů abecedy (v [1] jsou použity symboly 0 a 1, tedy počet symbolů je 2). Každý z dalších řádků představuje jeden příklad. Skládá se z informace o tom, zda je pozitivní (1), negativní (0) nebo není známo, zda jej automat přijímá, tedy jedná se o testovací data (-1). Následují délky řetězce a symboly řetězce oddělené bílými znaky. Délka řetězce je rovna počtu symbolů.

Příklad Abbadingo formátu trénovacích dat pro $I_+ = \{1000, 0100, 0010, 10111, 111101, 010000, 100000, 0001101, 0000101\}$ a $I_- = \{101, 0000, 1101, 00000, 00101, 010111, 1000111\}$:

```
16 2
1 4 1 0 0 0
1 4 0 1 0 0
1 4 0 0 1 0
1 5 1 0 1 1 1
1 6 1 1 1 1 0 1
1 6 0 1 0 0 0 0
1 6 1 0 0 0 0 0
1 7 0 0 0 1 1 0 1
1 7 0 0 0 0 1 0 1
0 3 1 0 1
0 4 0 0 0 0
0 4 1 1 0 1
0 5 0 0 0 0 0
0 5 0 0 1 0 1
0 6 0 1 0 1 1 1
0 7 1 0 0 0 1 1 1
```

Příklad Abbadingo formátu testovacích dat:

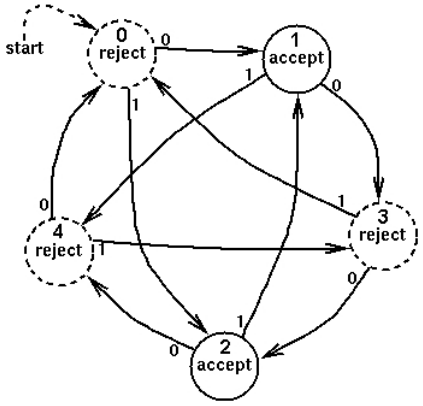
```
20 2
-1 7 0 0 1 1 0 0 0
-1 7 1 0 0 1 1 0 1
```

-1 7 1 0 1 0 1 0 1
-1 7 0 0 1 1 1 1 0
-1 7 0 1 0 0 1 0 1
-1 7 0 0 0 0 0 0 0
-1 6 1 0 1 0 0 1
-1 7 0 1 1 0 0 0 1
-1 7 1 0 0 0 0 1 0
-1 7 0 0 0 0 1 1 0
-1 5 0 0 0 1 0
-1 6 1 0 0 1 1 1
-1 6 0 0 0 0 0 1
-1 5 1 0 0 0 1
-1 7 1 1 1 0 1 1 1
-1 7 1 1 0 0 0 1 0
-1 7 1 0 0 0 0 0 1
-1 7 0 1 1 1 1 1 1
-1 7 1 0 0 1 1 1 1
-1 7 1 1 1 0 1 1 0

Abbadingo formát automatu obsahuje v hlavičce údaje o počtu stavů a velikosti abecedy (počtu symbolů). Každý další řádek reprezentuje jeden stav. Postupně obsahuje nezáporné celé číslo stavu, informaci o tom, zda je stav koncový a pro každý symbol z abecedy (nezáporné celé číslo) číslo stavu, do něhož se dostaneme, pokud je symbol na vstupu. Symboly se řadí podle abecedy vzestupně a berou se za sebou jdoucí čísla začínající od 0 (např. ve vzorcích se dvěma symboly to budou 0 a 1).

Příklad Abbadingo formátu automatu:

5 2
0 0 1 2
1 1 3 4
2 1 4 1
3 0 2 0
4 0 0 3



3.3 Trénovací a testovací data

V této podkapitole je podrobně vysvětlen způsob generování trénovacích a testovacích dat.

Aplikace generuje zadaný počet trénovacích dat obsahujících pozitivní i negativní vzorky, které se mohou opakovat. Pokud bychom tedy chtěli testovat algoritmus, který negativní vzorky nepotřebuje, musí se umět vypořádat s negativními vzorky na vstupu (například je ignorovat). Testovací data jsou omezena pouze podmínkou,

že vzorek nesmí být obsažen v trénovacích datech, tedy množiny trénovacích a testovacích dat musí být disjunktní. Jejich počet také zadává uživatel.

Při generování trénovacích a testovacích dat lze zvolit, zda trénovací data obsahují pozitivní a negativní vzorky náhodně nebo je polovina trénovacích dat pozitivní a polovina negativní. Poloviny se uplatní hlavně v případě, že daný jazyk je hustý, tedy cílový automat přijímá téměř všechna slova nad použitou abecedou a skoro všechny vzorky by bez použití dělení vzorků na poloviny byly pozitivní. Potom např. jednostavový automat od programu s algoritmem regulární inference přijímající všechny vzorky by z hlediska klasifikace testovacích dat dopadl velmi dobře (úspěšnost klasifikace by se blížila 100%).

Cílový automat také může přijímat úplně všechna slova nad danou abecedou, potom by neexistoval žádný negativní vzorek. V aplikaci se proto po nalezení dostatečného počtu pozitivních vzorků při generování dat na poloviny omezuje počet pokusů hledání negativních vzorků na 1000, po dosažení tohoto limitu je vypsána chyba a ukončen běh aplikace. Stejný limit je nastaven pro hledání pozitivních vzorků po nalezení dostatečného počtu negativních vzorků, kdyby automat nepřijímal žádná nebo téměř žádná slova. Dále se liší způsob generování dat:

- Pokud se jedná o binární vzorky, může uživatel vybrat metodu, která bere data náhodně z množiny $16n^2 - 1$ vzorků stejně jako v [11], kde n je počet stavů cílového automatu. Jejich délky leží mezi 0 a $2\log_2 n + 3$, jak je uvedeno v [11]. Tato metoda ale generuje delší vzorky s větší pravděpodobností, protože čím delší vzorek vezme, tím více existuje možností, jak bude vzorek vypadat a delších vzorků je tedy větší množství.

- Abychom se vyhnuli problému zmíněném na konci předchozího bodu, další metoda nejprve vybere délku vzorku a pak generuje symboly, dokud nedostane vzorek této délky. Všechny délky tedy mají stejnou pravděpodobnost a navíc lze použít i jiné než binární vzorky a tudíž otestovat automaty pracující s více než dvěma symboly, což předchozí metoda neumí. Jako počet generovaných vzorků se volí pro trénovací i testovací data druhá mocnina počtu stavů automatu normalizovaná požadovanou složitostí dat. Pokud aplikace testuje algoritmy třetí metodou popsanou v předchozí kapitole, jsou speciálně definovány požadované počty trénovacích a testovacích dat dle [11] (viz Tabulka 1).

3.4 Průběh testování

Tato podkapitola popisuje postupy přiložené aplikace implementující testování algoritmů regulární inference.

Aplikace umožňuje zadat na vstup automaty, na nichž má probíhat testování, třemi způsoby:

1. vygenerovat náhodné automaty, z nichž první má n stavů a každý další má o 4 stavy více (kde n je dělitelné 4 a určuje jej uživatel stejně jako počet automatů) nebo náhodné automaty pro testování metodou Abbadingo z podkapitoly 3.1,

které mají na rozdíl od předchozích předem dané počty stavů na 64, 128, 256 a 512 a mohou tedy být maximálně 4 (viz Tabulka 1),

2. načíst automaty ze vstupního souboru pevného formátu od uživatele,

3. načíst automaty pro Tomitovy jazyky, které jsou vypsány v Tabulce 2 zpracované podle [9] (počty stavů jsou upraveny kvůli použití Abbadingo formátu automatu pro binární vzorky).

Jazyk	**Popis (regulární výraz)**	**Počet stavů**
L1	0*	2
L2	(01)*	3
L3	Nemá lichý počet 1 a potom lichý počet 0	5
L4	Maximálně dvě po sobě následující 0	4
L5	Sudý počet 0 a sudý počet 1	4
L6	Počet 0 a počet 1 kongruentní modulo 3	3
L7	0*1*0*1*	5

Tabulka 2. Tomitovy jazyky.

Generování automatů je převzato z [11], tedy aplikace nejprve vytvoří orientovaný graf na $\frac{5}{4}n$ vrcholů stupně 2, náhodně vybere vrchol jako počáteční stav a vezme podgraf obsahující vrcholy dosažitelné z kořene. Pro každý vrchol náhodně určí, zda se bude nebo nebude jednat o přijímací stav. Generování se provádí tak dlouho, dokud maximální hloubka výsledného grafu není právě $2\log_2 n - 2$. Výsledkem popsaného postupu je automat pracující s binárními vzorky.

Po vytvoření automatů si aplikace načte informace o programech s algoritmy, které má testovat. Poté si vezme jeden automat a opakovaně se nagenerují trénovací a testovací data, na nichž se otestují všechny algoritmy. Protože testování probíhá pro všechny algoritmy na stejných datech, kontroluje se strukturální kompletnost trénovacích dat, pokud to požaduje alespoň jeden z testovaných algoritmů. Pokud skončí testování pro jeden automat, uloží se výsledky pro zvolenou testovací metodu a opakuje se stejný postup na ostatních automatech. Nakonec vypíše nebo vykreslí výsledky se statistikami ve formátu, jenž si uživatel vybere.

Kapitola 4

Závěr

4.1 Zhodnocení práce

Problém regulární inference je velmi zajímavý a užitečný. Má nemalý teoretický význam, ale také široké spektrum aplikací jako je např. identifikace sekvenčních procesů, rozpoznávání vzorů, zpracování řeči a přirozeného jazyka, exploratorní sekvenční analýza, umělá inteligence nebo data mining.

V aplikaci se podařilo implementovat několik metod pro testování algoritmů regulární inference a poskytnout tak vědeckým pracovníkům nástroj k usnadnění vývoje těchto algoritmů. Součástí aplikace je také několik programů s algoritmy regulární inference stažených z [2], aby bylo možné porovnat nové algoritmy regulární inference se schopnými a vyzkoušenými. Protože tyto algoritmy byly k dispozici pouze pro UNIXové systémy, jsou v rámci práce upraveny také pro MS Windows. Jako teoretický podklad aplikace slouží samotný text této práce, jehož součástí je v příloze i dokumentace k aplikaci.

Nejdůležitější a nejvíce používanou se pravděpodobně stane metoda testující schopnost algoritmu klasifikovat testovací data. Význam této metody spočívá v prezentaci schopností algoritmu zobecňovat na nová data. Zde však dále uživatel může zvolit způsob generování trénovacích a testovacích dat. Způsob použitý v [11], kterým je vybrán jeden vzorek z univerza je první možností. Ta očividně upřednostňuje delší vzorky před kratšími. Proto je druhou z možností nejdříve náhodně vybrána délka vzorku a pak je teprve vygenerován samotný vzorek této délky.

Celkový úspěch práce ukáže až její užití v praxi při vývoji algoritmů pro učení konečných automatů.

4.2 Možná vylepšení

Na každé práci je stále co zlepšovat. Proto vznikla tato kapitola, aby dala prostor k publikaci nápadů, které nebyly zrealizovány.

Určitě existuje více způsobů generování trénovacích a testovacích dat. Bylo by možno například nahlédnout na tuto problematiku z hlediska toho, s jakou pravděpodobností se automat vydá po jedné nebo druhé větvi místo pravděpodobnosti délky

vzorku nebo výběru vzorku z jejich dané množiny. Tyto metody totiž způsobí, že pokud jsou délky libovolných větví automatu vycházejících ze stejného stavu nerovnoměrné (na jedné z nich lze vygenerovat více vzorků), pak má jedna větev větší pravděpodobnost než druhá, že se při generování dat automat vydá právě po ní. To by řešila metoda, která by brala stejné pravděpodobnosti, že se automat vydá po jedné větvi (pokud je stav koncový, musíme to započítat jako další větev).

Jistě by se také daly zkoumat další metody testování algoritmů regulární inference a vymyslet tím další hlediska pohledu na kvalitu automatů, které tyto algoritmy vydávají na výstupu.

Dodatek A

Uživatelská dokumentace

A.1 Popis práce s aplikací

Aplikace slouží k testování algoritmů regulární inference vybranými testovacími metodami. Bližší popis najdeme v hlavní části textu této práce. Aplikace se spouští z příkazové řádky příkazem `TestAlgDFA` a komunikace s ní probíhá prostřednictvím textových konfiguračních souborů (vzorové soubory najdeme na přiloženém CD). Výstupy se pak ukládají rovněž do souborů. Nejprve si popíšeme vstupní konfigurační soubory, které je potřeba umístit do pracovního adresáře aplikace. Výstupní soubory najdeme rovněž v pracovním adresáři aplikace.

Hlavním konfiguračním souborem params.tadfa řídíme celý průběh testování. Jeho obsahem jsou hodnoty potřebných parametrů napsané pod sebou. Pokud nějakou hodnotu nepotřebujeme, vůbec se do souboru neuvádí. Následující seznam popisuje všechny parametry, které tento soubor obsahuje a jejich vzájemné závislosti (výrazem Px značíme parametr x, kde x je číslo parametru v seznamu):

1. Inicializovat random generátor ručně (1 = ano, 0 = ne). Aplikace používá generátor náhodných čísel. Jeho ruční nastavení na určitou hodnotu může být užitečné např. pokud hledáme v programu s algoritmem regulární inference nějakou chybu a potřebujeme jej nastavit opakovaně na stejnou hodnotu.

2. Druh automatu (1 = Tomitovy jazyky (v pracovním adresáři musí být soubor tomita.tadfa), 2 = ze souboru od uživatele, 3 = Abbadingo, n = z generátoru (n je počet stavů automatu dělitelný 4)).

3. Počet automatů (pokud P2 = 1, 3, 4). Jméno souboru (pokud P2 = 2).

4. Počet trénovacích dat (uvádí se jen pro P2 = 1, 2, 4). Je potřeba zadat rozumné množství, protože pokud zadáme mnoho (více než lze pro daný automat vygenerovat), mohla by aplikace spotřebovat všechny vzorky na generování trénovacích dat a v generování testovacích dat by nastala chyba (testovací data obsahují pouze vzorky, které nejsou v trénovacích datech). Pokud naopak zadáme příliš málo, může být učení méně úspěšné.

5. Počet testovacích dat (uvádí se jen pro P2 = 1, 2, 4).

6. Stejné pravděpodobnosti délek vzorků (1 = ano, 0 = ne, uvádí se jen pro P2 = 1, 2, 4). Pokud 1 z automatů nepracuje se 2 symboly, nelze zvolit 0.

7. Testovací metoda (1 = konzistence automatu s trénovacími daty, 2 = schopnost automatu klasifikovat testovací data, 3 = metoda Abbadingo, 4 = porovnání počtu stavů, 5 = porovnání MML; uvádí se jen pro P2 = 1, 2, 4).

8. Generovat trénovací data náhodně (1 = ano, 0 = ne). Pokud je zvoleno 0, bude vygenerována polovina pozitivních a polovina negativních vzorků.

9. Počet opakování generování dat pro automat. Jako výsledek se pak bere průměr z tohoto počtu opakování.

10. Jméno souboru pro uložení trénovacích dat (pro P13 = 1 najdeme soubory pod názvy jméno_automat_opakování_složitost, kde vše kromě jména jsou čísla příslušné položky).

11. Jméno souboru pro uložení testovacích dat (pro P13 = 1 najdeme soubory pod názvy jméno_automat_opakování_složitost, kde vše kromě jména jsou čísla příslušné položky).

12. Jméno souboru pro uložení automatu od programu s algoritmem (pro P14 = 1 najdeme soubory pod názvy jméno_automat_opakování_složitost_algoritmus_-nedeterminismus, kde nedeterminismus je číslo od 1 do 10 pro nedeterministické algoritmy, jinak je vždy rovno 1, vše ostatní kromě jména jsou čísla příslušné položky).

13. Ukládat trénovací a testovací data (1 = ano, 0 = ne).

14. Ukládat automaty z generátoru (pokud se automaty generují) a od programu s algoritmem (1 = ano, 0 = ne).

15. Vypisovat výstup do formátu csv (1 = ano, 0 = ne).

16. Vypisovat výstup jako tabulku do formátu tex (1 = ano, 0 = ne).

17. Vypisovat pro každý automat graf pro srovnání algoritmů (1 = ano, 0 = ne).

18. Vypisovat pro každý algoritmus graf pro srovnání automatů (1 = ano, 0 = ne).

19. Počet desetinných míst pro výpis procentuálních úspěšností ve výstupech.

20. Vypisovat na výstup také minima a maxima (1 = ano, 0 = ne). Pokud je P16 = 1, je maximální počet automatů 18 pro hodnotu 1 a 54 pro hodnotu 0. Pro P9 = 1 je průměr roven výsledku a minima i maxima se mu rovnají.

21. Číslo pro ruční inicializaci random generátoru (uvádí se jen pro P1 = 1).

22. Jméno souboru pro výstup do formátu csv (uvádí se jen pro P15 = 1).

23. Jméno souboru pro výstup do formátu tex (uvádí se jen pro P16 = 1).

24. Jméno souboru pro graf s porovnáním algoritmů (uvádí se jen pro P17 = 1).

25. Jméno souboru pro graf s porovnáním automatů (uvádí se jen pro P18 = 1).

26. Formát grafu (1 = png, 2 = gif, 3 = jpg; uvádí se jen pro P17 = 1 nebo P18 = 1).

27. Násobek základní velikosti grafu 100 x 75 (uvádí se jen pro P17 = 1 nebo P18 = 1).

28. Cesta k souboru s gnuplotem (absolutní či relativní; gnuplot\bin\pgnuplot znamená, že v pracovním adresáři aplikace se nachází adresář gnuplot, v něm adresář bin a v něm soubor pgnuplot.exe pro spuštění gnuplotu; uvádí se jen pro P17 = 1 nebo P18 = 1). Gnuplot je freeware (ke stažení např. na [7]).

29. Jméno souboru pro uložení automatu z generátoru (uvádí se jen pro P14 = 1, soubory najdeme pod názvy jméno_automat_opakování_složitost_algoritmus_-nedeterminismus, kde nedeterminismus je číslo od 1 do 10 pro nedeterministické algoritmy, jinak je vždy rovno 1, vše ostatní kromě jména jsou čísla příslušné položky).

Dále je potřeba do souboru algors.tadfa napsat informace o algoritmech, které chceme testovat. Každý řádek reprezentuje 1 algoritmus a obsahuje jméno programu s algoritmem (v MS Windows není potřeba udávat příponu .exe), informaci o tom, zda je algoritmus deterministický (1 = ano, 0 = ne), informaci o tom, zda algoritmus požaduje strukturální kompletnost trénovacích dat (1 = ano, 0 = ne) a další parametry pro program s algoritmem (např. přiložený algoritmus rlb potřebuje znát velikost okénka). Pokud chceme testovat automaty zadané uživatelem, je třeba vytvořit soubor, jenž obsahuje na prvním řádku počet automatů, dále následují jednotlivé automaty v Abbadingo formátu automatu. Jméno tohoto souboru sdělíme aplikaci v konfiguračním souboru params.tadfa popsaném výše.

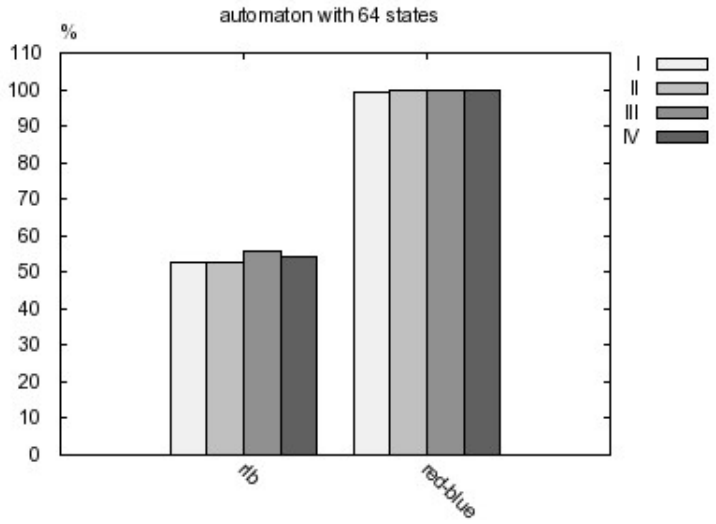
K aplikaci jsou přiloženy algoritmy red-blue, rlb a traxbar (podrobněji je popisuje poslední odstavec programátorské dokumentace). Každý testovaný program s algoritmem uvedený v souboru algors.tadfa musí být uložen v pracovním adresáři s aplikací v podobě spustitelného souboru.

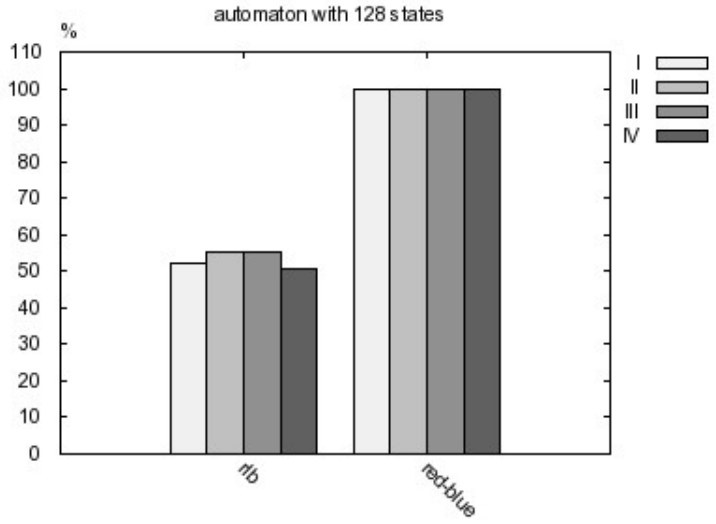
Výstup do formátu tex je reprezentován tabulkou, kde pro každý algoritmus najdeme výsledky ke všem automatům. Počet řádků tabulky pro algoritmus se tedy rovná počtu automatů a vypisovány jsou průměrné hodnoty. Pokud ale navíc vypisujeme minima a maxima, u každého automatu přibyde nad řádkem s průměry řádek s minimy a pod řádkem s průměry řádek s maximy.

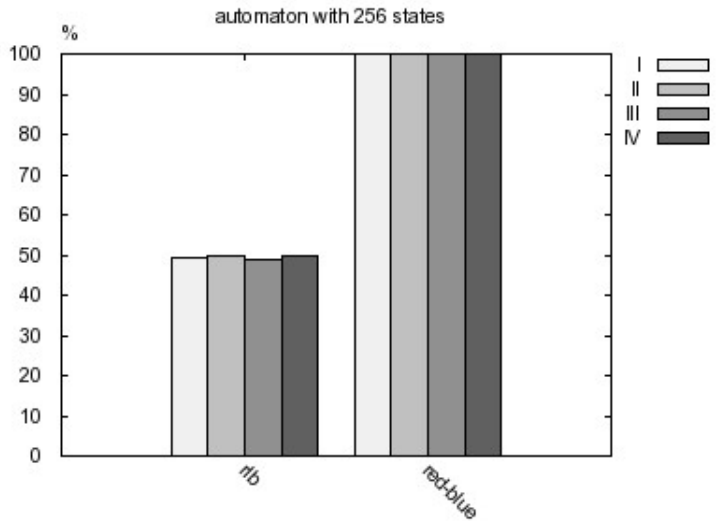
U výstupu v podobě grafu lze zvolit, zda se bude vykreslovat soubor pro každý automat a bude vždy obsahovat srovnání všech algoritmů nebo se bude vykreslovat soubor pro každý algoritmus a bude vždy obsahovat srovnání všech automatů. Srovnání je vyjádřeno procentuální úspěšností pro 4 složitosti trénovacích dat, kde horní hranice grafu je volena podle nejvyšší úspěšnosti, aby se dosáhlo co možná nejlepší čitelnosti. Je samozřejmě možné grafy vykreslovat i pro minima a maxima a ne jen pro průměry, potom jméno souboru s grafem končí „min“ resp. „max“.

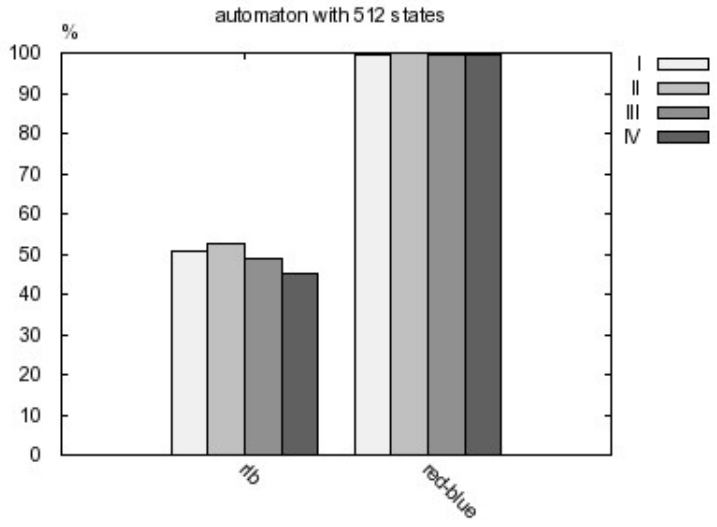
A.2 Příklad testování

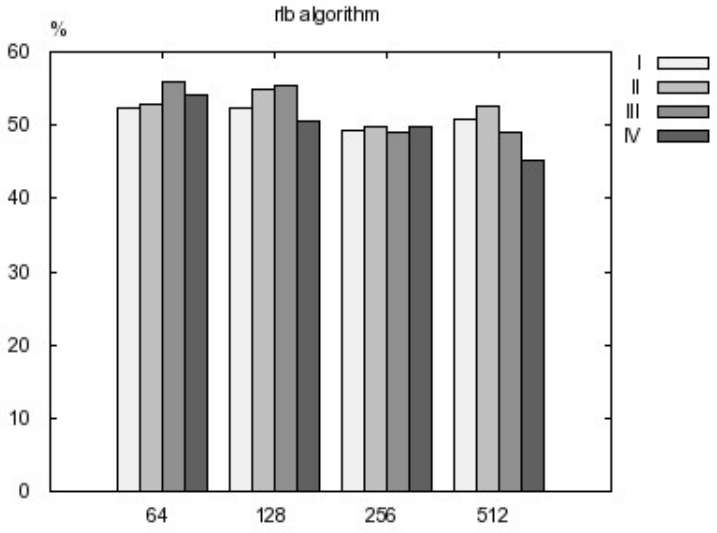
Následující grafy (4 pro porovnání klasifikačních schopností algoritmů na automatech majících 64, 128, 256 a 512 stavů, 2 pro znázornění změn klasifikačních schopností algoritmů s rostoucí velikostí automatu) a Tabulka 3 ukazují výsledky reálného příkladu testování metodou Abbadingo pro P1 = 0, P2 = 1, P8 = 1, P19 = 3, P20 = 0, P26 = 3 a P27 = 4. V tomto testování byly použity programy s algoritmy red-blue a rlb. Pro rlb bylo zvoleno okénko velikosti 10. Toto testování probíhá přesně podle soutěže Abbadingo [11], ale díky náhodnému generování automatů i dat a díky jinému vyhodnocení (v soutěži Abbadingo se bere jako výsledek procentuální úspěšnost, kde se pokus považuje za úspěšný, pokud je správně klasifikováno více než 99% testovacích vzorků, my bereme za výsledek průměr klasifikace testovacích dat ze všech opakování, což pro 1 opakování dává právě výsledek tohoto pokusu) se mohou výsledky lišit. Z grafů je vidět, že při zadaném nastavení algoritmus red-blue překonává algoritmus rlb, který nedopadl moc dobře.

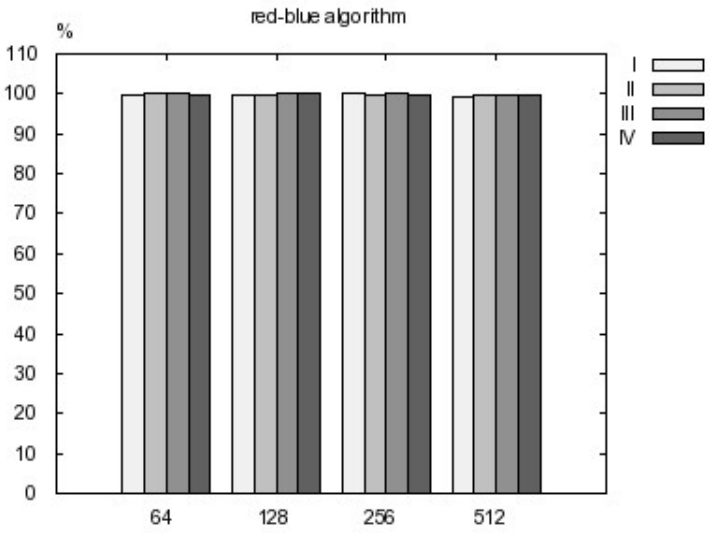





automaton with 256 states
%
100
90
80
70
60
50
40
30
20
10
0
rlb
red-blue
I
II
III
IV


automaton with 512 states
%
100
90
80
70
60
50
40
30
20
10
0
rlb
red-blue
I
II
III
IV


rlb algorithm
%
60
50
40
30
20
10
0
64
128
256
512
I
II
III
IV



target size	IV	III	II	I	algorithm
64	54.111	55.944	52.889	52.444	
128	50.500	55.444	55.000	52.389	rlb
256	49.833	49.056	49.778	49.389	
512	45.278	49.056	52.611	50.889	
64	99.833	100.000	100.000	99.500	
128	100.000	99.944	99.833	99.722	red-blue
256	99.889	99.944	99.889	99.944	
512	99.500	99.500	99.833	99.389	

Tabulka 3. Výsledek vzorového testování metodou Abbadingo.

Dodatek B

Programátorská dokumentace

Aplikace TestAlgDFA sloužící k testování kvality algoritmů regulární inference je napsána v jazyce C++ pro systémy MS Windows (32-bit) a systémy na bázi UNIXu. Zkompilována a testována byla v MS Windows XP Professional SP3 (EN) pod MS Visual Studio .NET 2005 Professional (EN) s nejnovějšími aktualizacemi k 13.7.2008 a v Linuxu Fedora 7 s jádrem 2.6.22.9-91.fc7 pod g++ (GCC) 4.1.2. Některé konstrukce (např. názvy funkcí) se mezi systémy MS Windows a systémy na bázi UNIXu liší, v hlavičkovém souboru TestAlgDFA.h jsou proto definována makra rozlišující mezi těmito systémy a podle nich je vždy zvolena konstrukce podporovaná v daném operačním systému.

Při kompilaci jsou potřeba kromě zdrojových souborů aplikace (TestAlgDFA.h, TestAlgDFA.cpp a MainProc.cpp) také hlavičkové soubory iostream, fstream, map, vector, deque, string, cstdlib, ctime, cmath, iomanip, stdexcept a fcntl.h. Kromě toho potřebujeme pod systémy MS Windows hlavičkové soubory io.h, sys/stat.h a process.h a pod systémy na bázi UNIXu hlavičkové soubory unistd.h a sys/wait.h. Pokud je některý z uvedených souborů v jiném umístění, než kompilátor očekává, musíme jej nahrát do adresáře se zdrojovými soubory aplikace nebo k němu nějakým způsobem zadat cestu.

Hlavní funkčnost aplikace obstarává funkce „main“, která používá další pomocné fukce definované v souboru MainProc.cpp a třídu „testovani“ deklarovanou v souboru TestAlgDFA.h a definovanou v souboru TestAlgDFA.cpp. Tato třída využívá další pomocné struktury a funkce deklarované a definované ve stejných souborech jako ona sama. Funkce „main“, třída „testovani“ a další zmíněné funkce a struktury budou v následujícím textu podrobně popsány.

Funkce „main“ nejprve pomocí funkce „nacti_parametry“ načte parametry zadané uživatelem v souboru params.tadfa (tento soubor je popsán v uživatelské dokumentaci) a inicializuje generátor náhodných čísel buď náhodně (podle času funkcí „time“) nebo číslem typu unsigned, které zadal uživatel v souboru params.tadfa. Dále vytvoří instanci třídy „testovani“ a do ní uloží požadované automaty. Pokud jsou automaty náhodně generovány, mohou být uloženy do souborů. Pak jsou ještě načteny algoritmy ze souboru algors.tadfa (tento soubor je popsán v uživatelské dokumentaci), alokuje se paměť pro parametry spuštění těchto algoritmů a může začít vlastní testování.

Testování probíhá pro každý automat ve 4 složitostech trénovacích dat a opakuje se vždy tolikrát, kolikrát určil uživatel. Pro 1 automat v 1 opakování jsou vždy pro 1 složitost otestovány všechny algoritmy na stejných trénovacích datech, která jsou pro ten účel vygenerována ve vytvořené instanci třídy „testovani“, zvolenou testovací metodou. Výsledky jsou ukládány také ve vytvořené instanci třídy „testovani“, potom jsou normalizovány počtem opakování a převedeny na procenta. Nakonec jsou vypsány výstupy v uživatelem zvolených formátech a je uvolněna paměť alokovaná pro parametry algoritmů. Funkce „main“ využívá tyto pomocné funkce:

- „nacti_parametry“
  Načítá parametry zadané do souboru params.tadfa uživatelem a vrací je ve svých parametrech.

- „nacti_algoritmy“
  Načítá algoritmy zadané do souboru algors.tadfa uživatelem a vrací je ve svém parametru.

- „preved_na_argv“
  Převede zadané parametry algoritmů na char** pro spuštění algoritmů.

- „nazev_data“
  Vytvoří název souboru pro trénovací nebo testovací data.

- „uloz_data“
  Uloží trénovací a testovací data do zadaných souborů.

- „nazev_aut“
  Vytvoří název souboru pro automat od algoritmu regulární inference.

- „nazev_gen“
  Vytvoří název souboru pro automat z generátoru.

- „vystup“
  Obstarává úpravu jmen souborů s grafy a volání vykreslení grafů. Grafy se vykreslují pro každý algoritmus (srovnání automatů) nebo pro každý automat (srovnání algoritmů).

Třída „testovani“ uchovává cílové automaty, automaty od algoritmu, trénovací i testovací data a výsledky testování. Používá k tomu struktury „stav“ pro uložení stavu automatu, „DFA“ pro uložení automatu, „vzorek“ pro uložení vzorku dat, „TTD“ pro uložení trénovacích nebo testovacích dat a „algoritmy“ pro uložení testovaných algoritmů a jejich parametrů. Zároveň poskytuje funkce pro práci s těmito daty. Nejdůležitější z nich si popíšeme (význam ostatních je zřejmý z komentářů):

- „gen_data“
  Generuje trénovací a testovací data. Využívá k tomu funkce „generuj_nahodne“, „generuj_napul“, „vzorek_rovnomerne“, „vzorek_nahodne“, „vrat_label“, „zkontroluj“, „je_napul“ a „kompletni“. Kontrola strukturální kompletnosti je

omezena na 1000 opakování, protože jinak by tato operace také nemusela nikdy skončit např. v případě, kdy pozitivních trénovacích vzorků je velmi málo.

- „run“
  Spustí program s algoritmem regulární inference a předá mu zadané parametry.
- „shoda_trenovaci“
  Metoda ověření konzistence automatu s trénovacími daty.
- „shoda_testovaci“
  Metoda testování schopnosti automatu klasifikovat testovací data.
- „pocet_stavu“
  Metoda porovnání počtu stavů.
- „shoda_mml“
  Metoda porovnání minimal message length.
- „vypis_csv“
  Vypisuje výstup do formátu csv.
- „vypis_tex“
  Vypisuje výstup jako tabulku do formátu tex.
- „vypis_graf“
  Vypisuje graf (png, gif nebo jpg). Využívá k tomu funkce „dat_gen“, „dat_alg“, „dat_aut“, „graf“, „horni_mez“ a „spust_graf“.
- „generuj_DFA“
  Vygeneruje konečný automat se zadaným počtem stavů. Využívá k tomu funkce „vytvor_graf“, „najdi_podgraf“, „prejmenuj_stavy“, „hloubka“ a „vytvor_automat“.
- „mml“
  Spočítá minimal message length pro zadaný automat. Využívá k tomu funkce „pocet_odchodu“ a „log2fakt“.

Součástí aplikace je rovněž několik algoritmů regulární inference, které byly staženy z [2]. Protože algoritmy z tohoto odkazu chodí jen pod systémy na bázi UNIXu, byly v rámci práce upraveny také pro systémy MS Windows. Potřebné úpravy vždy popisuje soubor readme.txt umístěný v adresáři s daným algoritmem. Jednou z nejdůležitějších úprav je nahrazení hlavičkového souboru sys/time.h souborem timeval.h, který byl stažen z [14]. Algoritmy se jmenují red-blue, rlb a traxbar. Algoritmus traxbar v některých případech vrací místo čísla stavu znak '?'. Při načítání automatu pak nastane chyba vstupu, protože při vstupních operacích probíhá kontrola datových typů (nebo obecně korektnosti vstupního proudu) pomocí funkce „kontrola“. Chybové hlášky jsou vypisovány funkcí „chyba“ a zároveň je program ukončen s kódem 1. Touto funkcí jsou tedy ošetřovány chyby, které zamezují dalšímu korektnímu běhu aplikace.

Literatura

[1] *Abbadingo One: DFA Learning Competition - DFAs: Languages and Learning* [online], citováno 24.06.2008, url: http://abbadingo.cs.unm.edu/dfa.html.

[2] *Abbadingo One: DFA Learning Competition - 4 DFA learning algorithms download* [on-line], citováno 24.06.2008, url: http://abbadingo.cs.may.ie/~lang/dfa-algorithms.tar.gz.

[3] Carrasco R. C., Oncina J.: *Learning stochastic regular grammar by means of a state merging method*, Grammatical Inference and Applications (ICGI'94), Springer, Berlin, Heidelberg, 1994, 139–152.

[4] Chytil M.: *Automaty a gramatiky*, SNTL, Praha, 1984, 18–96.

[5] Dupont P.: *Regular grammatical inference from positive and negative samples by genetic search: the GIG method*, Grammatical Inference and Applications (ICGI'94), Springer, Berlin, Heidelberg, 1994, 236–245.

[6] Dupont P., Miclet L., Vidal E.: *What is the search space of the regular inference?*, Grammatical Inference and Applications (ICGI'94), Springer, Berlin, Heidelberg, 1994, 25–37.

[7] *Gnuplot download - stránka pro stažení gnuplotu* [on-line], citováno 13.07.2008, url: http://www.gnuplot.info/download.html.

[8] Gold E. M.: *Complexity of Automaton Identification from Given Data*, Information and Control **37**, 1978, 302–320.

[9] Hingston P.: *A genetic algorithm for regular inference*, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Morgan Kaufmann, San Francisco, 2001, 1299–1306.

[10] Hoffmann P.: *Improving RPNI Algorithm Using Minimal Message Length*, Artificial Intelligence and Applications, Innsbruck, 2007, 378–383.

[11] Lang K. J., Pearlmutter B. A., Price R. A.: *Results of the Abbadingo One DFA learning competition and a new evidence-driven state merging algorithm*, Lecture Notes in Computer Science **1433**, 1998, 1–12.

[12] Oncina J., Garcia P.: *Inferring regular languages in polynomial update time*, Pattern Recognition and Image Analysis, World Scientific, 1992, 49–61.

[13] Trakhtenbrot B., Barzdin Y.: *Finite Automata: Behavior and Synthesis*, North Holland Publication Company, Amsterdam, 1973.

[14] Yongwei, Wu: *Hlavičkový soubor timeval.h* [on-line], citováno 13.07.2008, url: http://www.geocities.com/yongweiwu/timeval.h.txt.