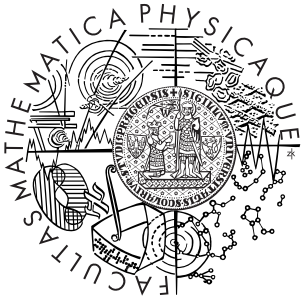
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Matěj Klonfar

Strojové učení formálních jazyků

Kabinet software a výuky informatiky

Vedoucí práce: RNDr. František Mráz, CSc.
Studijní program: Informatika, programování

2008

Chtěl bych na tomto místě poděkovat svému vedoucímu bakalářské práce za podnětné připomínky a především za trpělivost.

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

V Praze dne 29. května 2009 Matěj Klonfar

Obsah

Název práce: Strojové učení formálních jazyků
Autor: Matěj Klonfar
Katedra (ústav): Kabinet software a výuky informatiky
Vedoucí bakalářské práce: RNDr. František Mráz, CSc.
e-mail vedoucího: Frantisek.Mraz@mff.cuni.cz

Abstrakt: V předložené práci studuji úlohu strojového učení formálních jazyků. Úlohou práce je navrhnout a implementovat program umožňující studovat průběh a výsledky jednotlivých algoritmů na učení jazyků. Program podporuje učení z příkladů zadaných vstupní množinou pozitivních a negativních příkladů nebo učitelem, znajícím cílový jazyk. Program poskytuje nástroj na testování naučených jazyků. Hlavním cílem práce je navrhnout program s ohledem na snadné rozšiřování o další implementace algoritmů bez omezení na použité reprezentace naučených jazyků či složitosti těchto jazyků.

Klíčová slova: Strojové učení, formální jazyky, přepisovací pravidla, regulární jazyky

Title: Machine learning of formal languages
Author: Matěj Klonfar
Department: Department of Software and Computer Science Education
Supervisor: RNDr. František Mráz, CSc.
Supervisor's e-mail address: Frantisek.Mraz@mff.cuni.cz

Abstract: In the present work I study the task of machine learning of formal languages. The task of the work is to design and implement the program with anyone will be able to study the progress and results of algorithms for learning languages. With this program is possible to run algorithms for learning from examples. Examples could be sets of positive and negative examples of the result languages or sets expressed by teacher who knows the result language. The main task is to design easily extensible application with a view to inserting add-ins without no limits of used representation of learned languages or language complexity.

Keywords: Machine learning, formal languages, rewriting systems, regular languages

Kapitola 1

Úvod

1.1 Cíl práce

Cílem práce je vytvořit nástroj na učení formálních jazyků. Tento nástroj by měl umožňovat studium průběhu a výsledků jednotlivých algoritmů na učení jazyků. Nástroj bude primárně zaměřen na učení regulárních jazyků. Maximální důraz bude kladen na snadnou rozšiřitelnost o implementace algoritmů učící se i jiné třídy jazyků.

Nástroj bude obsahovat prostředky na testování naučených jazyků. Bude také podporovat učení s učitelem, kdy se žák (=algoritmus) učí hledaný jazyk na základě informací poskytnutých učitelem, znajícím cílový jazyk.

1.2 Motivace

Motivací této práce je poskytnout badateli v oblasti strojového učení formálních jazyků nástroj, pomocí něhož bude moci snadno studovat průběh algoritmů na strojové učení formálních jazyků na konkrétních příkladech včetně studia naučených jazyků.

1.3 Obsah dalších kapitol

Druhá kapitola přinese vysvětlení základních pojmů teorie formálních jazyků a konečných automatů. Seznámení se základními pojmy je nezbytné, pro správné pochopení dalších kapitol.

Třetí kapitola seznámí čtenáře s popisem analýzy a postupem při vývoji aplikace.

Čtvrtá kapitola rozebírá implementační detaily výsledného programu včetně popisu implementovaných algoritmů.

Závěrečná kapitola pak přinese závěrečné zhodnocení a nastíní směr dalšího vývoje.

Přílohou této práce jsou uživatelská a programátorská dokumentace vytvořené aplikace.

Součástí práce je přiložený CD nosič obsahující programovou část bakalářské práce a elektronickou podobu této práce.

Kapitola 2

Formální jazyky

2.1 Základní definice

Abecedou Σ rozumíme konečnou neprázdnou množinu znaků (písmen). Slovo nad abecedou Σ pak budeme chápat jako konečnou (i prázdnou) posloupnost písmen dané abecedy. Máme-li abecedu Σ, pak množinu všech slov označíme znakem Σ^* a množinu všech neprázdných slov znakem Σ^+. Prázdné slovo (tj. prázdná posloupnost znaků) bývá zpravidla označováno jako λ.

Formálním jazykem nazveme libovolnou množinu slov konečné délky nad danou abecedou. Příkladem formálního jazyka může být jazyk nad abecedou {a,b}, obsahující pouze slova, která neobsahují znak „b“.

Formální gramatika G je čtveřice (N, Σ, P, S),kde N je konečná množina neterminálních symbolů, Σ je neprázdná konečná množina terminálních symbolů (tj. abeceda jazyka), P je konečná množina pravidel ve tvaru $(\Sigma \bigcup N)^* N (\Sigma \bigcup N)^* \rightarrow (\Sigma \bigcup N)^*$ a S je počáteční neterminální symbol.

Řekneme, že slovo w se přímo přepíše na z ($w \Rightarrow z$), jestliže $\exists u, v, x, y \in (\Sigma \bigcup N)^*$ takové, že $w = xuy$ a $z = xvy$ a $(u \rightarrow v) \in P$. O slově z řekneme, že bylo odvozeno ze slova w, pokud existuje posloupnost přímých přepsání taková, že $w = x_0 \Rightarrow x_1 \Rightarrow ... \Rightarrow x_n = z$.

Formální jazyk definovaný pomocí formální gramatiky je množina všech slov obsahující pouze terminální symboly, která lze odvodit z počátečního symbolu za použití pravidel gramatiky. O takovém jazyku říkáme, že je generovaný danou gramatikou.

2.2 Chomského hierarchie

Chomského hierarchie dělí formální gramatiky podle tvaru příslušných gramatických pravidel do několika tříd.

gramatiky typu 3 (Regulární/pravé lineární jazyky) obsahují pouze pravidla ve tvaru: $X \to wY$ a $X \to w$, kde $X, Y \in N$ a $w \in \Sigma^*$.

gramatiky typu 2 (Bezkontextové jazyky) obsahují pouze pravidla ve tvaru: $X \to w$, kde $X \in N$ a $w \in (N \bigcup \Sigma)^*$.

gramatiky typu 1 (Kontextové jazyky) mohou obsahovat pouze pravidla ve tvaru: $\alpha X \beta \to \alpha w \beta$, kde $X \in N$, $\alpha, \beta \in (N \bigcup \Sigma)^*$ a $w \in (N \bigcup \Sigma)^+$ s výjimkou pravidla $S \to \lambda$, které se může být přítomno jen, pokud se symbol S nevyskytuje na pravé straně žádného pravidla.

gramatiky typu 0 (Rekurzivně spočetné jazyky) obsahují pravidla v obecné formě.

Chomského hierarchie definuje uspořádání tříd jazyků:

$$\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0,$$

kde $\mathcal{L}_0$,$\mathcal{L}_1$,$\mathcal{L}_2$,$\mathcal{L}_3$ jsou třídy jazyků generované gramatikami typu 0, typu 1, typu 2 a typu 3.

V práci se dále budeme zabývat jen regulárními jazyky a některými třídami jazyků bezkontextových (zejména jazyky definovanými pomocí systémů přepisovacích pravidel).

2.3 Regulární jazyky

Regulární jazyky představují třídu jazyků generovaných regulárními gramatikami. Regulární jazyky mohou být definovány kromě regulárních gramatik například regulárními výrazy nebo konečnými automaty. Nejčastěji je algoritmy na učení jazyků využívána reprezentace jazyka pomocí konečných automatů.

Konečný automat je definován jako pětice $(Q, \Sigma, \delta, S, F)$, kde Q je konečná množina stavů automatu, Σ je abeceda, δ je přechodová funkce automatu: $Q \times \Sigma \to Q$ u deterministických automatů, u nedeterministických automatů: $Q \times \Sigma \to P(Q)$, S je neprázdná množina počátečních stavů (u

deterministických automatu obsahuje $S = \{q_0\}$ právě jeden počáteční stav) a F je množina přijímajících stavů.

Automat je na počátku zpracování slova v definovaném počátečním stavu. V každém kroku přečte jeden symbol a přejde z aktuálního stavu do stavu, který je dán hodnoutou přechodové funkce odpovídající aktuálnímu stavu a přečtenému symbolu. Řekneme, že automat přijímá slovo, pokud po zpracování celého slova je automat v jednom z příjímajích stavů. Jazyk rozpoznáváný konečným automatem je pak množina všech jím příjímaných slov.

Třída jazyků rozpoznávaných deterministickými konečnými automaty je shodná s třídou jazyků rozpoznávaných nedeterministickými konečnými automaty.

2.4 Systémy přepisovacích pravidel [2]

Systémy přepisovacích pravidel jsou trojice (P, Σ, S), kde P je neprázdná konečná množina přepisovacích pravidel ve tvaru $\alpha \to \beta$, Σ je abeceda, $\alpha, \beta \in (\lambda + \$)\Sigma^*(\lambda + \&)$, kde \$ a & značí znaky začátku a konce slova a $S \in \Sigma$ je neredukovatelný řetězec. Znaky „\$“ a „&“ nelze použitím žádného přepisovacího pravidla přepsat či smazat. Použitím přepisovacího pravidla se nahradí nějaký výskyt levé strany přepisovacího pravidla ve slově jeho pravou stranou.

O systému přepisovacích pravidel řekneme, že je omezený pokud obsahuje pouze pravidla $(l \to r)$ těchto čtyř typů:

1. $l, r \in \$\Sigma^*$ (pravidlo přepisuje předpony řetězců),
2. $l, r \in \$\Sigma^*\&$ (pravidlo přepisuje celé řetězce),
3. $l, r \in \Sigma^*$ (pravidlo přepisuje části řetězců),
4. $l, r \in \Sigma^*\&$ (pravidlo přepisuje pouze přípony řetězců).

Nad slovy dále zavedeme délkově-lexikografické uspořádání, které je definováno takto: slovo a je menší než slovo b, pokud je buď kratší nebo lexikograficky menší. Toto uspořádání rozšíříme i na znaky začátku a konce slov takto: je-li $a < b$ pak $a < \$a < a\& < \$a\& < b$.

Řekneme, že přepisovací systém je hybridní pokud jeho přepisovací pravidla, která obsahují znak „\$“, mají pravou stranu pravidla délkově-lexikograficky menší než stranu levou a pravidla, která znak „\$“ neobsahují mají pravou stranu pravidla kratší než levou.

Hybridní systém přepisovacích pravidel zaručuje, že odvození každého slova bude konečně dlouhé. Při každé aplikaci přepisovacího pravidla je jeden výskyt levé strany daného pravidla nahrazen pravou stranou. O každém pravidlu platí, že jeho pravá strana je délkově-lexikograficky menší než strana levá, tedy i celý řetězec je po aplikaci pravidla délkově-lexikograficky menší. Stále však nezaručuje jedinečnost odvození, tedy že libovolná posloupnost aplikací přepisovacích pravidel vede vždy k témuž neredukovatelnému řetězci.

O systému přepisovacích pravidel řekneme, že je téměř se nepřekrývající, pokud pro všechna přepisovací pravidla $R_1 = l_1 \to r_1$ a $R_2 = l_2 \to r_2$ platí:

1. když $l_1 = l_2$, pak i $r_1 = r_2$
2. když $\exists u, v \in \Sigma^*, ul_1v = l_2, uv \neq \lambda$, pak $ur_1v = r_2$
3. když $\exists u, v \in \Sigma^*, l_1u = vl_2, 0 < |v| < |l_1|$, pak $r_1u = vr_2$

Jazyk rozpoznávaný systémem přepisovacích pravidel je množina všech slov nad abecedou Σ, z nichž lze odvodit daný neredukovatelný řetězec.

Třída jazyků rozpoznávaných pomocí omezených hybridních a téměř se nepřekrývajících systémů přepisovacích pravidel je podmnožinou třídy bezkontextových jazyků a obsahuje třídu regulárních jazyků.

Kapitola 3

Analýza a návrh

Cílem práce je vytvořit systém umožňující studovat průběh a výsledky jednotlivých algoritmů. V této analýze se tedy pokusím nalézt optimální model výsledné aplikace, který by umožnil do aplikace zahrnout různorodé algoritmy bez ohledu na to, jaké třídy jazyků jsou schopny se naučit a jaké prostředky k tomu používají.

3.1 Učení jazyků

Úloha učení formálních jazyků spočívá v hledání reprezentace netriviálního jazyka nad danou abecedou, vyhovujícího poskytnutým informacím (množinám pozitivních a negativních příkladů či informacím od učitele). Nalezený jazyk musí obsahovat všechny dané pozitivní příklady a žádný negativní. Žák (algoritmus) se musí pokusit pochytit i „smysl“ daných příkladů. Například z pozitivních příkladů „a“ a „aaa“ a negativního „aa“ by se měl dovtípit, že hledaný jazyk jsou slova liché délky.

Učení jazyka může probíhat za asistence učitele či bez ní. Při učení s učitelem získává algoritmus (žák) informace o hledaném jazyku z odpovědí učitele. Učitel, který zná cílový jazyk, odpovídá na algoritmem kladené dotazy. Při učení bez učitele pak musí algoritmus vystačit pouze se vstupními daty – množinami pozitivních příkladů (tj. příkladů slov z hledaného jazyka) a negativních příkladů (tj. slov mimo hledaný jazyk). Abecedu hledaného jazyka dostane algoritmus vždy od uživatele, respektive z načteného souboru se vstupními množinami příkladů.

V případě učení s učitelem se výpočet zastaví, pokud učitel shledá naučený jazyk ekvivalentní s jazykem cílovým, respektive nalezenou reprezentaci

ekvivalentní se známou reprezentací hledaného jazyka. Korektní algoritmus by tedy neměl skončit aniž by došel k hledanému jazyku. V případě učení jen ze vstupních příkladů je však nutné správnost nalezeného jazyka ověřit a případně upravit vstupní množiny příkladů.

3.2 Vstupní data

Vstupní data pro běh algoritmů tvoří specifikace abecedy a v případě algoritmů učících se pouze z množin příkladů i množiny pozitivních a negativních příkladů. Abeceda se skládá z písmen, což může být libovolný znak kromě znaku „-“, který je vyhrazen pro prázdné slovo. Množiny pozitivních a negativních příkladů jsou pak množiny slov nad touto abecedou.

Aplikace by měla umožňovat spouštět různé algoritmy nad těmi samými daty. Specifikace abecedy i množiny příkladů budou tedy uloženy v jednom zdrojovém souboru. Uživatel si bude moci otevřít tento soubor a nad ním pak spouštět jednotlivé algoritmy. Algoritmy si ze zdrojového souboru budou brát pouze informace, které potřebují ke svému běhu. Algoritmy, které nepracují s množinami příkladů si tedy ze zdrojového souboru budou brát pouze specifikaci abecedy.

Součástí aplikace by měly být nástroje na vytváření nových a editaci zdrojových souborů.

3.3 Role učitele

Učitel zná hledaný jazyk a je schopen odpovídat na některé otázky. Konkrétní učitel je úzce svázán s jednotlivými algoritmy, respektive skupinou algoritmů. Každý učitel podporuje jen několik typů dotazů, na které umí odpovídat. Učitel je tedy využitelný jen pro ty algoritmy, které pokládají jen učitelem podporované typy dotazů. Zároveň učitel odpovídá na základě znalosti reprezentace hledého jazyka a tudíž může spolupracovat jen s těmi algoritmy, které se jsou schopny naučit jen takto reprezentovatelné jazyky.

Každý učitel může být implementován ve dvou formách. Jako manuální učitel, který dotazy algoritmu (žáka) jen přeposílá uživateli a učícímu algoritmu vrací odpovědi uživatele, a automatický učitel, který odpovídá sám na základě znalosti hledaného jazyka, respektive některé jeho uživatelem předložené reprezentace.

Volba typu učitele nesmí ovlivnit průběh učícího algoritmu. Co nejmenší rozdíl v chování aplikace by měl být i z pohledu uživatele, kdy při volbě automatického učitele se jemu položené otázky, včetně jeho odpovědí, zobrazují stejně jako v případě manuálního učitele. Zároveň by měl mít uživatel možnost průběh učení s automatickým učitelem „krokovat“, čili moci testovat aktuální algoritmem naučený jazyk. U rozšiřujících implementací automatického učitele ovšem záleží na vlastní implementaci, kdy pozastaví běh algoritmu a umožní tím uživateli testovat poslední verzi naučeného jazyka.

Implementován bude základní typ učitele vhodný pro učení regulárních jazyků, který bude schopen odpovídat na otázky příslušnosti slova do cílového jazyka a otázky ekvivalence naučeného jazyka s cílovým, respektive nalezeného konečného automatu se známým automatem reprezentujícím hledaný jazyk. Samotná aplikace však nesmí být závislá na použití konkrétního učitele, jelikož by to omezilo její použitelnost pro některé algoritmy. Implementace učitelů budou tedy podobně jako implementace algoritmů zahrnuty do rozšiřujících balíčků.

3.4 Reprezentace naučeného jazyka

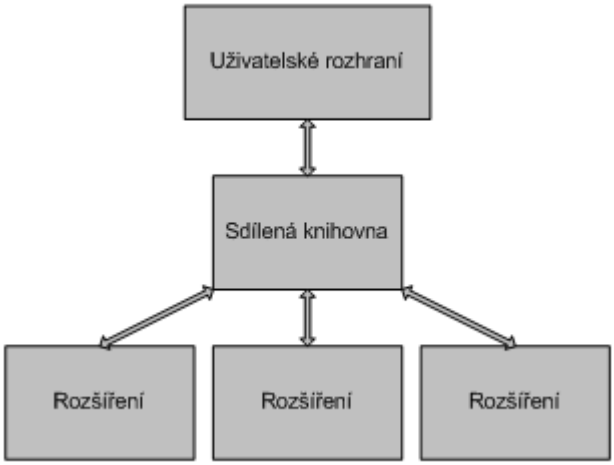
Zvolená reprezentace naučeného jazyka určuje jakou třídu jazyků je schopen se daný algoritmus naučit, případně jakým jazykům bude učitel rozumět. Cílem je vytvořit snadno rozšiřitelný systém a není možné proto zvolit a implementovat pouze několik podporovaných reprezentací. Podpora jen několika málo reprezentací jazyka by značně omezila výběr použitelných algoritmů. Pro snadnou tvorbu rozšíření bude aplikace poskytovat základní reprezentaci pro regulární jazyky – konečné automaty.

3.5 Rozšíření

Možnost snadno rozšiřovat funkčnost aplikace je jedním ze základních cílů této práce. Jak vyplývá z předešlých částí této kapitoly, nestačí možnost přidávat jen další algoritmy, ale je nutné umožnit přidávat i další potřebné struktury pro běh algoritmů (tj. učitele a reprezentace naučených jazyků). Rošíření budou do aplikace nahrávaná pomocí samostatných modulů ve formě dynamicky linnkovaných knihoven.

3.6 Rozdělení aplikace

Kvůli požadavku na snadnou rozšiřitelnost aplikace je nutné ji rozdělit do několika vrstev. Nejvyšší vrstvu bude tvořit jádro aplikace – uživatelské rozhraní a základní logika aplikace. Pod touto vrstvou bude mezivrstva implementovaná pomocí sdílené knihovny, která obsahuje definice základních rozhraní, jež musí jednotlivá rozšíření implementovat, a základní objekty poskytované aplikací. Nejnižší vrstvou pak bude vrstva jednotlivých rozšíření. Toto rozdělení umožňuje velmi komfortní způsob rozšiřování aplikace pomocí samostatně kompilovaných modulů, aniž by byl nutný zásah do samotné aplikace.



Obrázek 3.1: Rozdělení aplikace

3.7 Uživatelské rozhraní

Uživatelské rozhraní bude jednoduché a přehledné. Uživateli se během procesu učení budou průběžně zobrazovat aktuální verze naučeného jazyka. Pro samotnou prezentaci naučeného jazyka (respektive jeho reprezentace) byla zvolena velmi jednoduchá forma, kdy se reprezentace jazyka prezentuje uživateli v textové podobě. Informace poskytnuté v tomto textu a jeho formátování je pouze na dané reprezentaci.

K tomuto řešení bylo přistoupeno pro jeho jednoduchost a univerzálnost. Aplikace je primárně zaměřena na regulární a bezkontextové jazyky,

jejichž obvyklé reprezentace (konečné automaty, regulární výrazy, systémy přepisovacích pravidel, ...) lze snadno prezentovat v textové podobě.

I v případě, že je použit automatický učitel, by se měly jednotlivé položené dotazy zobrazovat v uživatelském rozhraní. Uživatelské rozhraní bude uchovávat a zobrazovat historii dotazů, a to jak v průběhu učení jazyka, tak i během jeho testování.

Uživatel musí mít přehled o průběhu běžícího algoritmu a možnost kdykoliv zastavit běžící algoritmus.

Kapitola 4

Implementace

Projekt byl realizován jako standardní aplikace pro operační systém Microsoft Windows. Je napsán v jazyce C++ pro platformu Microsoft .NET verze 2.0.

4.1 Implementované algoritmy

Implementováno bylo jen několik základních algoritmů na učení regulárních jazyků a algoritmus na učení systému přepisovacích pravidel. Snažil jsem se k implementaci vybrat několik zajímavých algoritmů, reprezentující odlišné přístupy. Ať už použitím odlišené reprezentace naučeného jazyka, využitím učitele či jen způsobem samotného učení.

4.1.1 Algoritmus RPNI [3]

Algoritmus RPNI (Regular Positive And Negative Inference) je příkladem algoritmu, který se učí hledaný jazyk pouze z poskytnutých vstupních informací - tj. abecedy jazyka a množin pozitivních a negativních příkladů. Algoritmus ke svému učení nevyužívá učitele. Naučený jazyk je v tomto algoritmu reprezentován pomocí deterministického konečného automatu. Algoritmus se učí prohledáváním prostoru deterministických konečných automatů o nejvýše N stavech, kde N je počet stavů výchozího prefixového automatu.

Algoritmus vychází z tzv. „prefixového“ automatu, což je deterministický konečný automat, který přijímá triviální jazyk obsahující právě slova, ze kterých je automat vytvořen. Tento „prefixový“ automat je vytvořen z množiny

pozitivních příkladů. V průběhu výpočtu algoritmus pracuje se dvěmi pracovními množinami stavů - množinou definitivních stavů „S“, která obsahuje stavy jež budou v cílovém automatu a množinou jejich sousedů „K“ v aktuální verzi automatu. Na počátku je za definitivní stav prohlášen vstupní stav automatu a do množiny „K“ jsou vloženi jeho sousedé.

V každé iteraci algoritmu, který běží dokud množina sousedů není prázdná, se pro jeden stav z množiny sousedů zkouší nalézt takový definitivní stav, že po jejich sloučení a obnovení determinicity automatu rekurzivním sloučením nedeterministicky dosažitelných stavů, bude automat dále přijímat všechny pozitivní a odmítat všechny negativní příklady. Není-li žádný takový definitivní stav nalezen, je tento stav prohlášen za definitivní a vložen do množiny S a jeho sousedé do K. Po ukončení algoritmu je tedy zaručeno, že vyhovuje zadaným množinám příkladů.

Algoritmus 1 rmerge(A,p,q)

/∗ sluč stavy p a q ∗/
$A \leftarrow merge(A, p, q)$
/∗ dokud existují z jednoho stavu dvě hrany se stejným písmenem, sluč stavy do kterých vedou ∗/
while $\exists a \in \Sigma : p, q \in \delta_A(r, a), p \neq q$ **do**
  $rmerge(A, p, q)$
end while

4.1.2 Algoritmus L^* [1]

Algoritmus L^* je podobně jako algoritmus RPNI zástupcem algoritmů učících se regulární jazyky reprezentované pomocí deterministických konečných automatů. Na rozdíl od algoritmu RPNI však využívá učitele a na základě jeho odpovědí postupně konstruuje deterministický konečný automat reprezentující cílový jazyk. Učitel musí být schopen odpovídat na otázky příslušnosti slova do hledaného jazyka (tzv. „membership query“) a otázky ekvivalence naučeného jazyka s cílovým (tzv. „equivalence query“). V každé iteraci algoritmus provede sérii „membership“ dotazů završenou jedním „ekvivalence“ dotazem. Výpočet algoritmu končí schválením jím naučeného jazyka učitelem - tj. kladnou odpovědí na ekvivalenci naučeného jazyka a jazyka cílového.

Algoritmus pracuje s deterministickým konečným automatem. Automat zde ale není popsán stejně jako například v algoritmu RPNI (tj. přímý popis

Algoritmus 2 Algoritmus RPNI

```
/* S - množina definitivních stavů */
/* K - množina sousedů definitivních stavů */

A ← PTA(S+) /* vytvoř prefixovy automat z pozitivních příkladů */
S ← q0 /* do S zařaď počáteční stav */
K ← {δ(q0, a), a ∈ Σ} /* do K zařaď sousedy počátečního stavu */
while K ≠ ∅ do
  vyber q ∈ K
  if ∃p ∈ S : L(rmerge(A, p, q)) ⋂ S− = ∅ then
    A ← rmerge(A, p, q)
  else
    /* nelze sloučit, zařaď stav do S */
    S ← S ∪ {q}
  end if
  /* doplň do K sousední vrcholy vrcholů z S */
  K ← {δ(s, a), a ∈ Σ, s ∈ S} − S
end while
```

stavů a hran), ale algoritmus udržuje tzv. „observation table“, která popisuje konečný automat pomocí charakteristických řetězců jednotlivých stavů. Tabulka je horizontálně rozdělena na oblast stavů (množina S) a oblast hran z nich vedoucích (množina E). Sloupce tabulky pak odpovídají příponám, které rozlišují jednotlivé stavy. Políčko tabulky vyjadřuje, zda slovo vzniklé zřetězením označení řádku a sloupce patří do cílového jazyka či nikoliv. Jsou-li dva řádky tabulky shodné až na označení řádku, pak odpovídají stejnému stavu.

Aby tabulka byla korektním popisem deterministického konečného automatu, musí splňovat podmínky uzavřenosti a konzistence. Podmínka uzavřenosti zajišťuje, že každá hrana vede do některého ze známých stavů, tedy že pro každý řádek tabulky z oblasti hran existuje ekvivalentní řádek z oblasti stavů. Podmínka konzistence pak testuje, zda pro každé dva shodné řádky v oblasti stavů platí, že všechny hrany z nich vedoucí a označené stejným písmenem abecedy vedou do shodného stavu (jejich řádky jsou schodné). Vyplní-li algoritmus svoji tabulku tak, že splňuje obě tyto podmínky, pak jí reprezentovaný konečný automat je prohlášen za kandidáta na hledaný automat a algoritmus pošle učiteli dotaz typu „equivalence query“.

Porušuje-li tabulka podmínku uzavřenosti, znamená to, že pro některý z řádků z oblasti hran neexistuje odpovídající řádek v oblasti stavů. Oblast stavů je tedy rozšířena o tento řádek a oblast hran doplněna o hrany z něj vedoucí. Porušuje-li tabulka podmínku konzistence, musí se přidat nový sloupec označený příponou rozlišující dva ekvivalentní stavy.

Tabulka je vyplňována pomocí učitelových odpovědí na dotazy typu „membership query“. Algoritmus je implementován, tak že se na každé slovo (políčko) ptá pouze jednou, i když se v tabulce může vyskytovat vícekrát. A před položením dotazu nahlíží nejprve do vstupních množin příkladů.

Deterministický konečný automat je pak z uzavřené a konzistentní tabulky definován takto: stavy automatu jsou řádky tabulky z oblasti stavů, počáteční stav je stav odpovídající řádku λ, přijímající stavy jsou ty stavy, pro které jsou políčka tabulky odpovídající charakteristickému řetězci stavu rovné 1, a přechodová funkce je $\delta(row(s), a) = row(s \cdot a)$.

4.1.3 Algoritmus LA [4]

Algoritmus LA je opět algoritmus učící se pomocí učitele. Stejně jako algoritmus L^* využívá pouze dotazy typu „membership query“ a „equivalence query“. Algoritmus však pracuje s nedeterministickým konečným automatem.

Algoritmus vychází z triviálního automatu, který má pouze jediný, a to nepřijímající stav. Pokud je učitelova odpověď na poslední dotaz typu „equivalence query“ pozitivní protipříklad (tedy slovo by mělo být přijímáno, ale není), pak je postupně pro každou předponu daného protipříkladu vytvořen nový stav (pokud takový stav už neexistuje). Nově vzniklé stavy jsou propojeny hranami na úplný automat (tj. z každého stavu vede pro každé písmeno abecedy hrana do všech stavů) a pak podobným způsobem připojeny k původnímu automatu (tj. z každého původního stavu jsou vytvořeny opět hrany pro všechny písmena abecedy do všech nových stavů). Stav s charakteristickým řetězcem daného protipříkladu je označen jako přijímající. Je-li učitelova odpověď negativní protipříklad, je pro tento protipříklad nalezena nějaká cesta do přijímajícího stavu. Pak postupně pro každou hranu této cesty od počátku je položen dotaz typu „membership query“ na slovo vzniklé zřetězením charakteristického řetězce výchozího stavu hrany a zbytku slova (přípony protipříkladu počínající n-tým znakem při zkoumání n-té hrany cesty). Při kladné odpovědi se algoritmus posune o hranu dál, při negativní je hrana z automatu vypuštěna.

Algoritmus 3 Algoritmus L*

/* $S, E \in \Sigma^*$ */
/* S – množina stavů */
/* E – množina přípon */
/* T – tabulka $(S \cup S \cdot \Sigma) \cdot E \to \{0, 1\}$ */

$S, E \leftarrow \{\lambda\}$
vyplň tabulku T
repeat
  while (S, E, T) není uzavřená a konzistentní **do**
    if (S, E, T) není konzistentní **then**
      najdi $s_1, s_2 \in S, a \in \Sigma, e \in E : T(s_1 \cdot E) = T(s_2 \cdot E)$ and $T(s_1 \cdot a \cdot e) \neq T(s_2 \cdot a \cdot e)$
      vlož $a \cdot e$ to E
      doplň T
    end if
    if (S, E, T) není uzavřená **then**
      najdi $s_1 \in S, a \in \Sigma : \ \forall s \in S \ T(s_1 \cdot a \cdot E) \neq T(s \cdot E)$
      vlož $s_1 \cdot a$ to S
      doplň T
    end if
  end while
  /* (S, E, T) je uzavřená a konzistentní */
  /* z tabulky vytvoř automat a navrhni ho učiteli */
  $A(Q, \Sigma, \delta, \{q_0\}, F) \leftarrow (S, E, T)$
  navrhni A učiteli
  if učitel odpověděl protipříkladem t **then**
    vlož t a všechny jeho předpony do S
    doplň T
  end if
until učitel nepotvrdil navrhovaný A
return A

Algoritmus 4 Procedura diag(Trans($M, p\prime, w\prime$))

/* $Trans(M, p\prime, w\prime)$ – posloupnost navazujících hran ze stavu $p\prime$ pro slovo $w\prime$ */
$Trans(M, p\prime, w\prime) : p\prime \rightarrow^a p \rightarrow^w q$ where $w\prime = aw$ $(a \in \Sigma, w \in \Sigma^*, p = [y])$
if $w = a \in \Sigma$ **then**
  /* jediná hrana */
  return $r : p\prime \rightarrow^a q$
else
  if $yw \in L$ **then**
    call diag(M, p, w)
  else
    return $r : p\prime \rightarrow^a p$
  end if
end if

Algoritmus 5 Algoritmus LA

/* počáteční automat rozpoznávající prázdný jazyk */
$M = (p_0, \Sigma, \emptyset, \{p_0\}, \emptyset)$, where $p_0 = [\lambda]$
while učitel neschválí M **do**
  /* učitel poskytl protipříklad */
  if $w \notin L(M)$ **then**
    /* pozitivní protipříklad */
    /* z předpon protipříkladu vytvoř nové stavy */
    $Q \leftarrow Q \cup Q(w)$
    /* vytvoř nové hrany */
    $\delta \leftarrow \delta \cup \delta_{new}$
    /* stav odpovídající w přijímající */
    $F \leftarrow F \cup \{[w]\}$
  else
    /* negativní protipříklad */
    /* najdi cesty pro w vedoucí do přijímajících stavů */
    najdi $Trans(M, p_0, w)$
    /* najdi a odstraň nevyhovující hrany */
    $T \leftarrow Trans(M, p_0, w)$
    $\delta \leftarrow \delta - diag(T)$
  end if
end while

4.1.4 Algoritmus LARS [2]

Algoritmus LARS (Learning algorithm for rewriting systems) je, jako jediný z implementovaných algoritmů, schopen se naučit některé třídy bezkontextových jazyků. Jako reprezentaci naučeného jazyka používá systém přepisovacích pravidel. Algoritmus se učí pouze na základě znalosti vstupních množin pozitivních a negativních příkladů a vrací omezený, hybridní a téměř se nepřekrývající systém přepisovacích pravidel a neredukovatelný řetězec.

Algoritmus pracuje se čtyřmi pracovními množinami. Množina R obsahuje naučená přepisovací pravidla, množiny I_+ a I_- obsahují normalizované řetězce pozitivních a negativních příkladů. Normalizace znamená, že původní slova jsou převedena na neredukovatelné řetězce použitím aktuálně naučených přepisovacích pravidel. Množina F obsahuje všechny podřetězce pozitivních příkladů setříděné podle délkově-lexikografického uspořádání včetně znaků počátku a konce slova. Z této množiny se vybírají hledaná přepisovací pravidla. Algoritmus se učí hybridní systém, tudíž stačí prohledávat jen polovinu kartézského součinu $F \times F$.

Algoritmus postupně skládá z množiny $F \times F$ přepisovací pravidla a zkouší je zahrnout do již naučeného systému pravidel. Uvažovány jsou jen ta pravidla, která mohou být aplikována na alespoň jeden řetězec z množiny I_+. Algoritmus konstruuje hybridní omezený téměř se nepřekrývající systém přepisovacích pravidel. Obě strany pravidla musí být tedy stejného typu a nové pravidlo nesmí porušovat podmínku téměř se nepřekrývajícího systému. Pravidlo je zařazeno do naučeného systému pokud splňuje všechny tyto podmínky a zároveň nově normalizované množiny I_+ a I_- mají prázdný průnik. Po prohledání celého prostoru možných přepisovacích pravidel algoritmus končí. Minimální řetězec (ve smyslu délkově-lexikografického uspořádání) z množiny I_+ je pak hledaný neredukovatelný řetězec naučeného systému. Pro všechny ostatní zbylé řetězce v I_+ je vytvořeno pravidlo přepisující tyto řetězce na neredukovatelný řetězec.

4.2 Modul „LEAR“

Tento modul představuje samotnou aplikaci. Jak vyplývá z předchozí kapitoly poskytuje uživatelské rozhraní a vytváří základní logiku aplikace.

Aplikace je implementována jako dvouvláknová, kdy v hlavním vlákně běží uživatelské rozhraní a ve druhém se spouští jednotlivé algoritmy. Ob-

Algoritmus 6 Algoritmus LARS

$R \leftarrow \emptyset, I_+ \leftarrow S_+, I_- \leftarrow S_-$
/* F – uspořádaná množina všech podřetězců pozitivních příkladů */
$F \leftarrow sort_{\preceq}\{v : \exists u, w \in \bar{\Sigma}^*, uvw \in I_+\}$
for $i = 1$ to $|F|$ **do**
  if přínosné$(F[i], I_+)$ **then**
    /* může přepsat alespoň jeden řetězec z I_+ */
    for $i = 0$ to $i - 1$ **do**
      /* hledaná pravá strana musí být délkově-lexikograficky menší */
      if type$(F[i])$ = type$(F[j])$ **then**
        /* obě strany jsou stejného typu – můžou tvořit přepisovací pravidlo */
        $S \leftarrow R \cup \{F[i] \rightarrow F[j]\}$
        if $ANo(S)$ **then**
          /* S je skoro se nepřekrývající systém pravidel */
          /* najdi neredukovatelné řetězce z pozitivních i negativních příkladů */
          $E_+ \leftarrow$ normalize(I_+, S)
          $E_- \leftarrow$ normalize(I_-, S)
          if $E_+ \cap E_- = \emptyset$ **then**
            /* přidej nové pravidlo */
            $R \leftarrow S, I_+ \leftarrow E_+, I_- \leftarrow E_-$
          end if
        end if
      end if
    end for
  end if
end for
/* vyber nejmenší z neredukovatelných řetězců a doplň pravidla na převod ostatních */
$e \leftarrow min_{\preceq} I_+$
for all $w \in I_+$ **do**
  if $w \neq e$ **then**
    $R \leftarrow R \cup \{w \rightarrow e\}$
  end if
end for
return $< R, e >$

jekt k synchronizaci obou vláken poskytuje hlavní třída aplikace. Jedná se o instanci třídy AutoResetEvent. Volání dotazů manuálního učitele jsou z pohledu vlákna algoritmu blokující, jelikož je nutné počkat na uživatelovu odpověď, jež čekající vlákno odblokuje.

Základní logika aplikace je vytvářena pomocí hlavní třídy aplikace „CMainClass“.

Tento modul dále obsahuje implementace obou forem základního typů učitele, který je schopen akceptovat regulární jazyk ve formě deterministického konečného automatu a odpovídat na otázky typu „membership query“ a „equivalence query“.

4.3 Ošetření chybových stavů

Jelikož je díky možnostem rozšíření aplikace možné do programu přidávat rozšíření, aniž bychom měli nějakou záruku jeho kvality a stability, je nutné implementovat robustní mechanismus ošetření chybových stavů.

Ošetření chybových stavů hrozících pádem aplikace bylo provedeno na nejvyšší úrovni, tedy v modulu „LEAR“ ve třídě představující hlavní formulář aplikace („CMainForm“) a v hlavní třídě aplikace („CMainClass“), pomocí mechanismu výjimek. Veškeré zachycené výjimky jsou předávány hlavnímu formuláři, který informace o nastalém chybovém stavu zobrazí uživateli. Vlákno algoritmu, který způsobil výjimku, je zrušeno.

4.4 Správa rozšíření

Správa jednotlivých rozšíření je implementována třídou „CDLLManager“. Rozšíření se načítají vždy při startu aplikace z pevně daného umístění (podadresář „plugins“). Není potřeba tedy udržovat žádnou konfiguraci. Jednotlivá rozšíření (objekty) jsou v programu identifikovány modulem, ze kterého pochází a názvem objektu. Je tedy možné aby aplikace obsahovala různé verze téhož objektu. Pouze v případě rozšíření o novou reprezentaci jazyka je nutné zajistit, aby každá reprezentace měla svůj unikátní název, pomocí něhož se identifikuje datový soubor. Jako platná rozšíření jsou brány ty moduly, které obsahují implementaci alespoň jednoho z rošiřujících rozhraní.

4.5 Modul „dll_interface“

Tento modul definuje základní typy pro funkci aplikace. Důležité jsou zejména definice společných rozhraní, jejichž implementací je možné rozšiřovat aplikaci. Podrobná dokumentace je v přiložené „Programátorské dokumentaci“ Jedná se o tato rozhraní:

ILanguage, definující základní metody pro reprezentace jazyků. Jedná se zejména o umožnění základních funkcí aplikace bez ohledu na používanou reprezentaci. Tedy o funkci testování a manipulaci se zdrojovým souborem.

ITeacher, které definuje základní metody pro funkci učitelů. Zejména to jsou definice metod jednotlivých typů dotazů. Definovány jsou dva typy. Dotaz, kdy předmětem dotazu je slovo, a dotaz, kdy předmětem dotazu je reprezentace jazyka. Konkrétní reprezentace dotazu je vždy na konkrétním učiteli.

IAlgorithm, toto rozhraní definuje základní metody pro obsluhu algoritmů. Rozhraní definuje pouze metody na spuštění algoritmu a oznámení výsledků. Neklade tedy na jednotlivé algoritmy žádné omezující požadavky.

Kapitola 5

Závěr

5.1 Naplnění cíle práce

Výsledkem této bakalářské práce je použitelný nástroj umožňující nahlédnout do průběhu učení jednotlivých algoritmů. Tento nástroj je univerzálně použitelný bez ohledu na přístup algoritmů k samotnému učení či použité reprezentace jazyka. Dokážu si představit i začlenění implementací algoritmů využívajících zcela odlišných prostředků než jsou implementované konečné automaty a systémy přepisovacích pravidel.

Efektivita aplikace záleží na implementacích jak samotných algoritmů, tak i automatických učitelů. Standardně poskytovaný automatický učitel byl implementován spíše za účelem demonstrace této možnosti, než s ohledem na maximální efektivitu.

5.2 Další vývoj

Další vývoj by měl spočívat zejména v postupném doplňování aplikace o implementace dalších algoritmů.

Literatura

[1] Angluin D.: *Learning regular sets from queries and counterexamples*, Information and Computation, 75:87–106, November 1987.

[2] R. Eyraud and C. de la Higuera and J. C. Janodet: *Representing Languages by Learnable Rewrtitng Systems* In: Proceedings of the 7th International Colloquium on Grammatical Inference, LNAI 3264, pages 139–150, Springer, 2004.

[3] J. Oncina and P. García:*Inferring regular languages in polynomial update time* In N. Pérez de la Blance, A. Sanfeliu, and E. Vidal, editors, Pattern Recognition and Image Analysis, volume 1 of Series in Machine Perception and Artificial Intelligence, pages 49–61, World Scientific, 1992.

[4] Yokomori T.: *Learning non deterministic finit automata from queries and counterexamples*, Machine Intelligence 13. Furukawa, Michie & Muggleton editors, Oxford University Press, 1993.

Příloha A

Uživatelská příručka

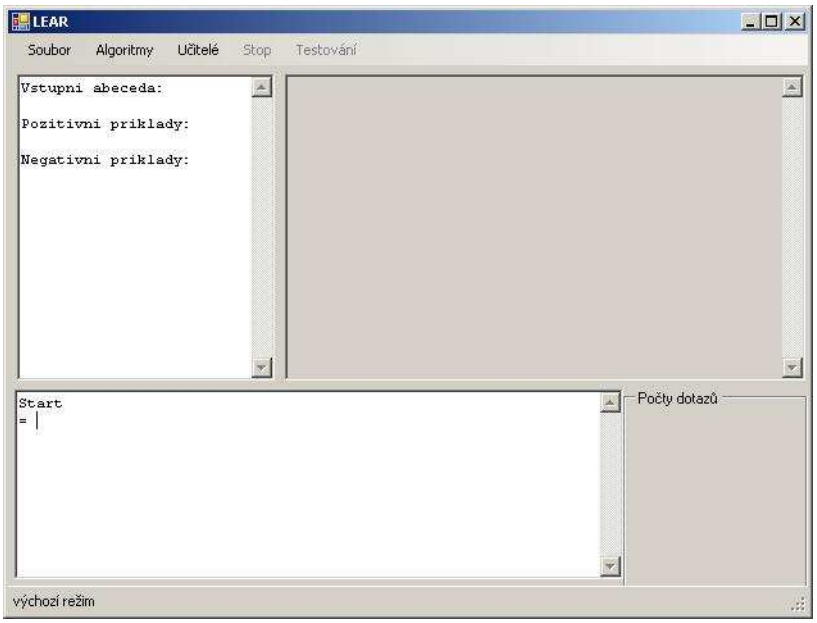
Program pro svůj běh vyžaduje nainstalovaný Microsoft .NET framework alespoň verze 2.0 SP1, který je volně ke stažení na stránkách www.microsoft.com.

Program umožňuje sledovat průběh algoritmů na strojové učení formálních jazyků. Primárně program obsahuje algoritmy na učení regulárních jazyků (algoritmy RPNI, L* a LA) a algoritmus na učení jazyků reprezentovatelných systémem přepisovacích pravidel (LARS). Program umožňuje běh algoritmů učících se jak pouze na základě informací obsažených ve vstupní množině příkladů, tak i učících se ve spolupráci s učitelem. Program obsahuje dvě implementace učitelů schopné odpovídat na dotazy příslušnosti slova do hledaného jazyka a ekvivalence aktuálního naučeného jazyka s jazykem cílovým. Manuální učitel pouze předává jemu položené dotazy uživateli a algoritmům vrací uživatelovu odpověď. Automatický učitel odpovídá algoritmům na základě znalosti konečného automatu reprezentující hledaný jazyk.

Hlavní okno aplikace (viz obrázek A.1) je rozděleno do tří oken: okno zobrazující otevřenou množinu příkladů vlevo nahoře, okno zobrazující aktuální naučený či načtený jazyk vpravo nahoře a konzolové okno ke komunikaci s uživatelem dole. V pravém dolním rohu se zobrazuje jednoduchá statistika položených dotazů učiteli.

Aplikace pracuje ve třech režimech:

výchozí režim Výchozí režim aplikace, ve kterém se nachází po spuštění dokud není spuštěn některý z algoritmů nebo načten dříve uložený jazyk. Uživatel v tomto režimu nemůže zapisovat do konzolového okna. Ostatní funkce aplikace jsou přístupné.



Obrázek A.1: Hlavní okno po spuštění aplikace

režim testování V tomto režimu může uživatel testovat aktuální jazyk. Aplikace do tohoto režimu vstoupí po ukončení běhu algoritmu, případně načtením dříve uloženého jazyka. Při testování uživatel zapíše do konzolového okna slovo jež testuje a aplikace odpovídá „Ano“ nebo „Ne“ podle příslušnosti testovaného slova do jazyka.

režim učení V tomto režimu se aplikace nachází v průběhu výpočtu některého algoritmu. Uživatel, v případě že algoritmus spolupracuje s manuálním učitelem, v konzolovém okně odpovídá na dotazy algoritmu. Je-li aplikace v režimu učení je možné pozastavit běh algoritmu a přepnout aplikaci do režimu testování a zpět stisknutím položky hlavního menu „Testování“ respektive „Odpověď“.

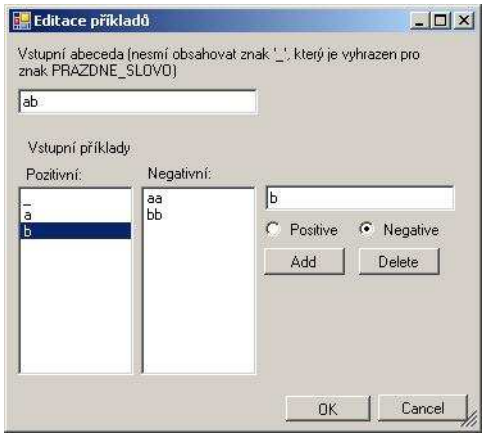
Aktuální režim aplikace je zobrazen v levém dolním rohu hlavního okna aplikace. Běžící algoritmus je možné kdykoli v průběhu výpočtu ukončit stisknutím tlačítka „Stop“.

A.1 Práce s aplikací

Aplikace po svém spuštění založí prázdnou množinu příkladů. Uživatel tak může ihned po spuštění pracovat s načtenými algoritmy. Pro úspěšné spuštění některého z algoritmů je však nutné zadat minimálně abecedu hledaného jazyka případně i množiny pozitivních a negativních příkladů. Je-li potřeba zadat abecedu či množiny příkladů bude o to uživatel požádan před samotným spuštěním algoritmu. Běžící algoritmus je možné kdykoli zastavit pomocí tlačítka „Stop“.

A.1.1 Editace množin příkladů

Aplikace umožňuje jednoduchou správu množin vstupních příkladů (záložka menu Soubor - Příklady). Uživatel může vytvářet nové a editovat stávající. Množina vstupních příkladů se skládá ze specifikace abecedy, pozitivních příkladů a negativních příkladů. Abeceda může obsahovat libovolné znaky (písmena) kromě znaku „_“, který se v aplikaci používá jako zástupný znak za prázdné slovo (λ). Jeden příklad nemůže být jak mezi pozitivními tak mezi negativními příklady. Příklady, kromě příkladu „_“, mohou obsahovat jen písmena abecedy. Jednotlivé příklady lze přidávat a mazat. Přesouvání příkladu mezi množinami pozitivních a negativních příkladů je možné pouze jeho odstraněním a opětovným přidáním.



Obrázek A.2: Editace množin příkladů

A.1.2 Formát zobrazení jazyka

Aplikace rozeznává dva typy jazyků. Jazyky reprezentované konečným automatem a jazyky reprezentované systémem přepisovacích pravidel. Další typy jazyků lze přidat pomocí rozšiřujících balíčků.

Formát zobrazení konečného automatu

Konečný automat se zobrazuje jako výpis jednotlivých stavů automatu a hran z nich vedoucích. Nejdříve je na samostatném řádku popsán stav: příznak „=>“ jde-li o počáteční stav nebo příznak „<=“ jde-li o přijímající stav, číselné označení stavu a znovu příznak 1 nebo 0, zda je stav přijímající nebo ne. Pod tímto řádkem jsou pak vypsány všechny hrany vedoucí z tohoto stavu. Vždy písmeno hrany a označení cílového stavu hrany.

Na obrázku A.3 je zobrazen automat o 2 stavech. Stav s označením 0 je jediný počáteční stav automatu a také jediný přijímající stav. Ze stavu 0 vedou obě hrany do stavu 1 a z něj pak hrana s písmenem „a“ vede zpět do stavu 0, zatímco hrana s písmenem „b“ zůstává ve stavu 1.

Formát zobrazení systému přepisovacích pravidel

Systém přepisovacích pravidel tvoří neredukovatelný řetězec a seznam přepisovacích pravidel, která jsou vypsána ve tvaru $leva\_strana => prava\_strana$.

A.1.3 Výběr učitele

Pro algoritmy, které se učí ve spolupráci s učitelem, je možné před jejich spuštěním vybrat typ učitele, který bude použit. V záložce menu „Učitele“ jsou k dispozici všechny načtené typy učitelů.

Implementovány byly dva typy učitelů: automatický a manuální. Oba umí odpovídat pouze na dotazy příslušnosti daného slova do cílového jazyka a na dotazy ekvivalence naučeného jazyka s jazykem cílovým. Ve výchozím nastavení je vybrán „Manuální učitel“.

Manuální učitel

Manuální učitel zprostředkovává komunikaci mezi běžícím algoritmem a uživatelem. Do konzolového okna jsou vypisovány dotazy algoritmu a uživatel na ně odpovídá. V kterémkoliv okamžiku, kdy už je znám nějaký naučený jazyk, je možné pomocí položky hlavního menu „Testování“ přerušit běh

algoritmu a přepnout aplikaci do režimu testování. Z režimu testování (viz oddíl A.1.4)) se stiskem položky hlavního menu „Odpověď“ aplikace přepne zpět do režimu učení a znovu vypíše poslední položenou otázku.

Manuální učitel vypisuje otázky algoritmu do konzolového okna včetně stručné nápovědy, jak odpovědět. Způsob odpovědi se liší podle typu položeného dotazu. Učitel rozpozná tři typy odpovědi:

odpověď „Ano“ je možná u obou typů dotazů. Odpoví se pouhým stisknutím klávesy „Enter“. Není potřeba do konzolového okna nic psát.

odpověď „Ne“ je možná pouze u dotazů na příslušnost slova do hledaného jazyka. Odpoví se tak, že do konzolového okna zapíše libovolný text a stiskne klávesa „Enter“.

protipříklad je možné dát pouze u dotazů na ekvivalenci naučeného jazyka s hledaným. Protipříklad se zapíše stejně jako předchozí typ odpovědi. Protipříklad smí obsahovat pouze písmena z abecedy hledaného jazyka. Prázdné slovo („λ“) se zapíše jako „_“.

Na obrázku A.3 je ukázka běhu algoritmu L* s manuálním učitelem. V konzolovém okně jsou vidět položené dotazy spolu s odpovědmi.

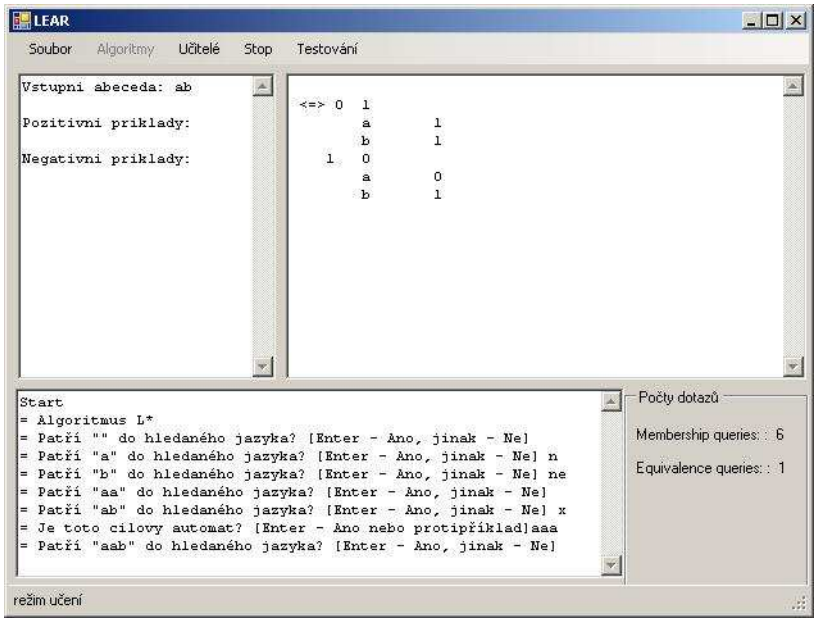
Automatický učitel

Automatický učitel potřebuje ke svému běhu znát cílový jazyk. Při spouštění algoritmu je uživatel dotázán na zdrojový soubor pro automatického učitele. Učitel umí pracovat pouze s jazyky reprezentovanými pomocí konečných automatů. Nazdaří-li se načtení souboru nebo soubor neobsahuje popis konečného automatu, algoritmus se nespustí.

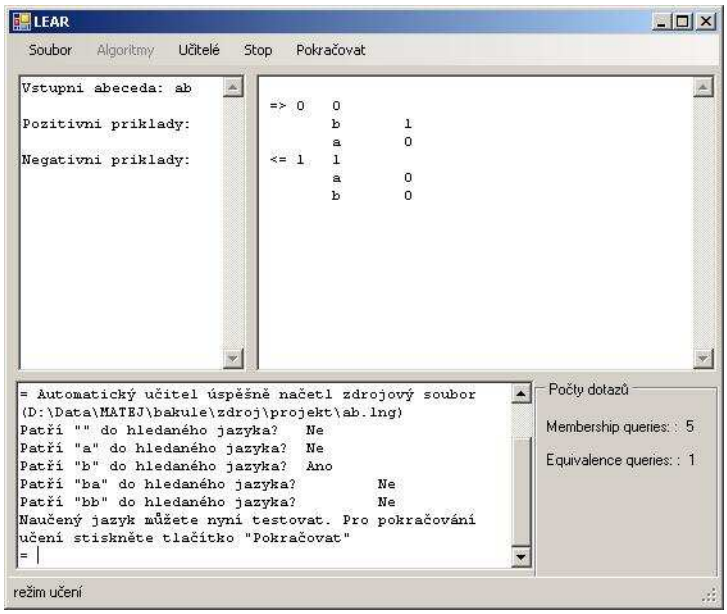
Učení s automatickým učitelem je rozfázováno do etap mezi dotazy na ekvivalenci naučeného jazyka s hledaným jazykem. Vždy když algoritmus položí tento dotaz, běh algoritmu se přeruší a uživatel může testovat aktuální verzi naučeného jazyka. Pro pokračování běhu algoritmu je nutné stisknout položku hlavního menu „Pokračovat“.

A.1.4 Testování jazyka

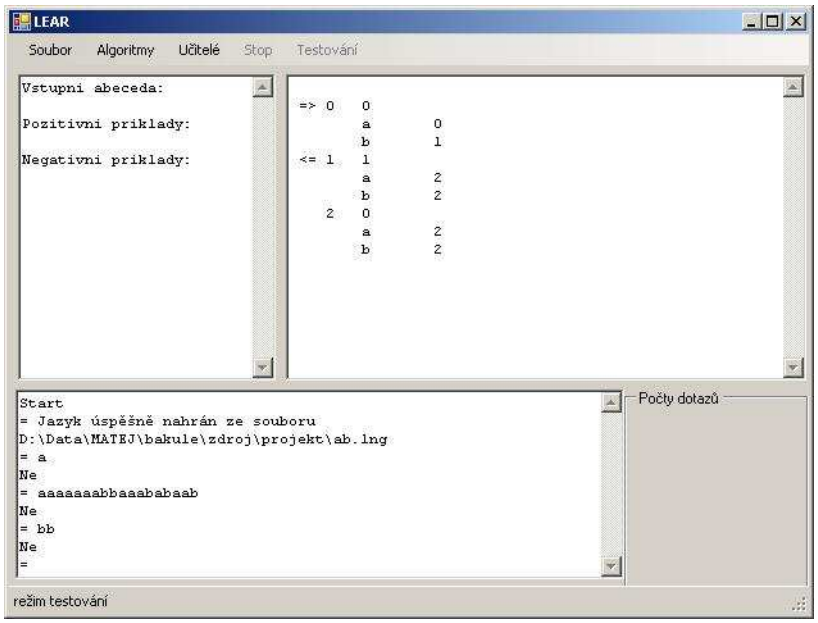
Při testování jazyka uživatel zadává do konzolového okna slova a aplikace odpovídá „Ano“ či „Ne“, podle toho jestli slovo patří do aktuálně načteného jazyka či nikoliv. Je možné jen, je-li aplikace v příslušném režimu.



Obrázek A.3: Ukázka běhu algoritmu L* s manuálním učitelem



Obrázek A.4: Ukázka běhu algoritmu L* s automatickým učitelem



Obrázek A.5: Ukázka testování jazyka

A.1.5 Spouštění algoritmů

Algoritmy se spouští z menu ze záložky „Algoritmy“. Pro úspěšné spuštění algoritmu musí otevřená množina příkladů obsahovat neprázdnou abecedu hledaného jazyka a pracuje-li algoritmus i s množinami pozitivních a negativních příkladů, pak i alespoň jeden příklad. Není-li splněná jedna z těchto podmínek, je před samotným spuštěním algoritmu uživatel požádán o doplnění chybějících informací a poté vyvoláno okno „Editace množin příkladů“. Pakliže ani poté nejsou splněny podmínky pro běh algoritmu, spouštění selže a do konzolového okna se vypíše jaká podmínka pro běh algoritmu nebyla splněna.

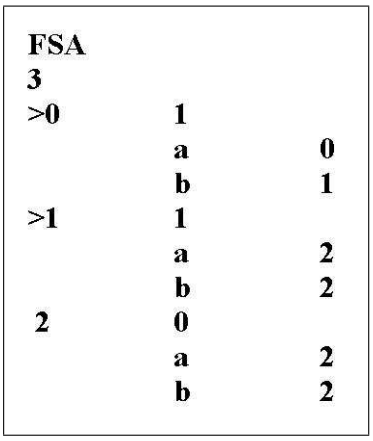
A.1.6 Ukládání a načítání jazyků

Aplikace umožňuje ukládat naučené jazyky do souborů a znovu je načítat za účelem testování či jako zdrojové soubory pro automatické učitele (záložka menu Soubor - Jazyky). Formát souboru je dán typem jím reprezentovaného jazyka. Aplikace momentálně podporuje formáty konečného automatu a systému přepisovacích pravidel. Soubory mají standartně příponu „lng“.

Konečné automaty

Formát uložení konečných automatů je velmi přehledný i pro lidské oko. V zásadě se jedná o seznam stavů a hran z nich vedoucích. Každý stav může být označen jako počáteční (znak „>“ před označením stavu, jinak řádek začíná mezerou) a jako přijímající (příznak 1/0 za označením stavu). Hrana je vypsána jako písmeno abecedy a cílový stav. Konečné automaty se identifikují podle řetězce „FSA“.

FSA
$< pocet\ stavu >$
seznam stavů:
$[> \mid] < id\_stavu > \quad < prijimajici >$
seznam hran vychazejících ze stavu:
$< pismeno > \quad < id\_ciloveho\_stavu >$



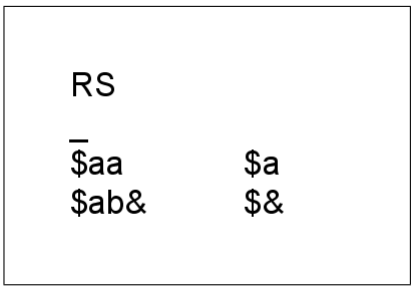
Obrázek A.6: Příklad uložení nedeterministického konečného automatu s dvěmi počátečními a přijímajícími stavy.

Systémy přepisovacích pravidel

Systémy přepisovacích pravidel mají identifikaci „RS“. Formát jejich uložení je velmi prostý. Nejdříve je na samostatném řádku neredkuvatelný řetězec systému a na dalších řídcích pak vždy dvojice levé a pravé strany přepisovacího pravidla oddělená tabulátorem:

RS
$< neredukovatelny\_retezec >$

seznam pravidel ve tvaru:
$< leva\_strana >< tab >< prava\_strana >$



Obrázek A.7: Příklad uložení systému přepisovacích pravidel s neredukovatelným řetězcem „“ a pravidly $\$aa \rightarrow \$a$ a $\$ab\& \rightarrow \$\&$

A.2 Instalace rozšíření

Aplikace byla vytvořena s důrazem na sandnou rozšiřitelnost. Aplikaci je možné rošířít o nové implementace algoritmů, typy učitelů a reprezentace jazyka. K instalaci rozšiřujícího balíčku jej stačí vykopírovat do podadresáře „plugins“ v adresáři aplikace. Rozšíření se nahrávají vždy při startu aplikace. Načteny jsou vždy všechny rozšiřující moduly v daném adresáři, není tedy nutné udržovat konfiguraci. Úspěšné načtení rozšiřujícího algoritmu a učitele se projeví zobrazením položky s jménem rošíření v příslušné záložce hlavního menu.

Aplikace nekontroluje závislosti mezi různými rozšířeními. Přidává-li uživatel například implementaci nového algoritmu, musí zároveň zajistit, aby byla v aplikaci přítomna i podpora pro algoritmem používanou reprezentaci jazyka.

Příloha B

Programová dokumentace

B.1 Struktura projektu

Program je napsán v jazyce C++ pro platformu Microsoft .NET 2.0. Program je řešen jako vícevláknová aplikace, kde v jednom vlákně běží samotný program (tedy uživatelské rozhraní) a jednotlivé algoritmy se pouští ve vlákně druhém. Program je rozdělen na několik logických částí. Jednou je samotný hlavní program, který obsahuje uživatelské rozhraní, hlavní třídu programu („CMainClass“) a mechanismus správy rozšíření („CDLLManager“). Další částí je knihovna „dll_interface“, která vytváří mezivrstvu mezi jádrem aplikace a jednotlivými rozšířeními. Knihovna definuje rozhraní, která jednotlivá rozšíření implementují a obsahuje i jejich základní částečnou implementaci. Jedná se o rozhraní „IAlgorithm“,„ITeacher“,„ILanguage“ a „IMainForm“ a objekty „CAlgorithm“,„CAbstractTeacher“,„CPriklady“ a implementaci základní reprezentace jazyka - konečných automatů, „CAutomat“. Poslední částí pak jsou jednotlivé rozšiřující moduly.

B.2 Důležité části kódu

B.2.1 Definice důležitých rozhraní

IMainClass

Rozhraní „IMainClass“ definuje základní metody pro spolupráci hlavního formuláře s učitelem či algoritmem. Definuje zejména tzv. oznamovací funkce, které aktualizují uživatelské rozhraní.

`void WriteText(String^ str);` funkce vypíše daný řetěze do „konzolového“ okna.

`void WriteLanguage(ILanguage^ lng);` funkce aktualizuje výpis aktuálního jazyka.

`void setAction(TAction a);` funkce nastaví režim aplikace (může být „Odpověď“, „Testování“ nebo „Výchozí režim“).

`void EndOfAlgorithm();` funkce oznámí konec algoritmu. Funkce je volána z metody RunAlgorithm objektu CAlgorithm.

`void presentStat(int count, array<String^ >^ names, array<int>^ stat);` prezentuje uživateli aktuální statistiku učitele o počtech různých typů dotazů.

`String^ getResponse();` funkce vrací uživatelovu odpověď. Funkci používá manuální učitel.

IAlgorithm;

Rozhraní „IAlgorithm“ definuje základní metody algoritmů.

`String^ getAlgorithmName();` funkce vrací jedinečný název algoritmu pro položku v menu hlavního formuláře a rozpoznání správného rozšíření.

```
void setTeacher(ITeacher^ teach);
virtual void setForm(IMainForm^ f);
void setExamples(CPriklady^ ex);
ILanguage^ getLanguage();
```

`void RunAlgorithm(void);` funkce spouští samotný algoritmus.

`bool TeacherWanted();` funkce vrací, zda algoritmus využívá učitele.

Z tohoto rozhraní odvozená abstraktní třída „CAlgorithm“ dále implementuje základní mechanismus běhu algoritmu a pro něj potřebné metody a vlastnosti.

`virtual void Run()=0;` virtuální metoda implementující samotný algoritmus.

`virtual void RunAlgorithm();` implementace mechanismu spouštění algoritmu a oznámení jeho úspěšného konce.

ITeacher

Rozhraní „ITeacher“ definuje základní metody pro objekt učitele.

`String^ getName();` funkce vracející název učitele.

`bool ask(String^ query);` dotazu na vztah slova a hledaného jazyka.

`bool ask(ILanguage^ lang, String^ *res);` dotaz na vztah aktuálního a hledaného jazyka. V parametru res se vrací případný protipříklad.

```
void setFormReference(IMainForm^ imf);
void setExamplesRef(CPriklady^ ex);
IMainForm^ getForm();
```

ILanguage

Rozhraní „ILanguage“ definuje základní metody pro konkrétní reprezentace jazyka. Definuje metody pro načtení, uložení, testování a prezentaci do hlavního formuláře.

`String^ getIDString();` vrací identifikační řetězec, podle kterého se rozeznávají jednotlivé reprezentace jazyka.

`void Save(String^ filename);`

`int Load(String^ filename);` formát uložení je závislý na zvolené reprezentaci, na prvním řádku souboru však vždy musí být identifikační řetězec reprezentace.

`int Test(String^ word);` funkce vrací zda je dané slovo prvkem reprezentovaného jazyka či nikoliv.

`String^ toText();` funkce vytvoří řetězcovou reprezentaci jazyka na text. Používá se k vypsání na formulář.

B.2.2 Popis základních objektů

Třída CMainClass

Třída CMainCLass představuje hlavní třídu programu. Uchovává důležité informace pro běh programu a vykonává funkce uživatelského prostředí. Jediná instance této třídy se vytváří při startu hlavního formuláře (událost „OnLoad“). Při své inicializace vytváří instanci třídy „CDLLManager“, která je zodpovědná za načtení všech relevantních rozšíření.

Třída CDLLManager

Třída nalezne a spravuje nalezená rozšíření. Třída na žádost vrací instance objektů jednotlivých rozšíření. Důležité metody:

`CDLLManager(void);` prochází všechny dll knihovny v podadresáři „plugins“ a hledá rozšíření.

Funkce vracející instance objektů rozšíření. Indexy `pos` jsou definovány pořadím ve kterém byly jednotlivá rozšíření nalezena:

```
CAlgorithm^ getAlgorithmObject(int pos);
ILanguage^ getLanguageObject(String^ str);
CAbstractTeacher^ getTeacherObject(int pos);
```

Třída CAutomat

Třída představuje objekt konečného automatu. Třída implementuje rozhraní ILanguage. Definuje identifikační řetězec konečného automatu „FSA“. Implementace nevyžaduje, aby při testování jím definovaného jazyka, automat obsahoval z každého stavu hranu pro každé písmeno abecedy. Kromě metod na manipulaci s konečným automatem obsahuje tyto důležité metody:

`int SjednotStavy(int i, int j);` funkce sjednotí stavy s indexem i a j, pokud existují. Typicky se touto operací z automatu stává nedeterministický.

`CAutomat^ RecursiveMerge();` dokud existuje více stavů do nichž vede hrana se stejným znakem z jednoho stavu, tak je sjednocuje. Výstupem je nový deterministický konečný automat.

`void DoplnHrany(String^ alphabet);` podle dané abecedy doplní do automatu chybějící hrany. Přidávané hrany směruje do nového tzv. „trash“ stavu.

`bool findExample(String^ *e);` funkce hledá řetězec „vedoucí“ některého z přijímajících stavů.

`List<int>^ NajdiCestu(String^ slovo);` funkce se pokouší najít cestu z počátečního stavu do nějakého přijímajícího stavu určenou daným slovem.

`CAutomat^ minimalizuj();` funkce minimalizuje konečný automat. Očekává, že automat je deterministický.

`CAutomat^ determinizuj();` determinizuje konečný automat.

Třída CPriklady

Třída CPriklady představuje reprezentaci vstupní množiny pozitivních a negativních příkladů a specifikaci abecedy hledaného jazyka. Třída pracuje se soubory výše specifikovaného formátu. Třída obsahuje nástroj (dialog) na editaci načteného souboru. Důležité metody:

`void Edit();` spustí dialog umožnující editaci načteného souboru. Soubor a načtená data jsou aktualizována při úspěšném ukončení dialogu.

`void PresentExamples(TextBox^ tb);` vypíše načtená data do daného TextBoxu. Slouží k prezentaci na hlavní formulář.

`int findExample(String^ e);` funkce slouží k eliminaci zbytečných „membership“ dotazů. Vrací příznak zda je dané slovo prvkem pozitivních (+1) nebo negativních (-1) příkladů. Není-li slovo obsažené v ani jedné množině příkladů funkce vrací 0.

B.3 Formáty souborů

B.3.1 Vstupní množiny příkladů

Vstupní množiny příkladů mohou být ukládány do textových souborů. Na prvním řádku soubor je vypsána kompletní abeceda jazyka. Na každém dalším řádku je pak vždy jeden z příkladů uvozený znakem „+“, pokud se jedná o pozitivní příklad, nebo znakem „-“, pokud se jedná o příklad negativní.

$< abeceda >$
seznam příkladu:
$[+-] < priklad >$

B.3.2 Reprezentace jazyků

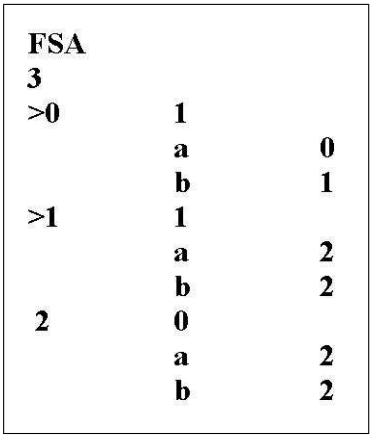
Jazyky jsou ukládány do souborů. Formát souborů je daný zvolenou reprezentací. Všechny však na prvním řádku musí obsahovat identifikační řetězec, podle kterého aplikace pozná o kterou reprezentaci se jedná. Na prvním řádku zároveň nesmí být nic jiného.

Konečné automaty

Formát uložení konečných automatů je velmi přehledný i pro lidské oko. V zásadě se jedná o seznam stavů a hran z nich vedoucích. Každý stav může být označen jako počáteční (znak „>“ před označením stavu, jinak řádek začíná mezerou) a jako přijímající (příznak 1/0 za označením stavu). Hrana je vypsána jako písmeno abecedy a cílový stav. Konečné automaty se identifikují podle řetězce „FSA“.

FSA
$< pocet\ stavu >$
seznam stavů:
$[> \mid] < id\_stavu > \quad < prijimajici >$

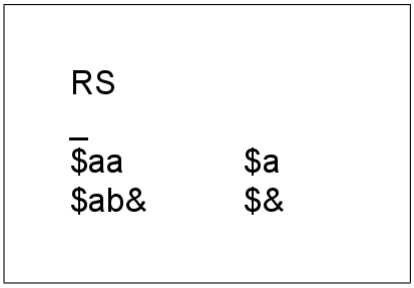
seznam hran vychazejících ze stavu:
$< pismeno >$ $< id\_ciloveho\_stavu >$



Obrázek B.1: Příklad uložení nedeterministického konečného automatu s dvěmi počátečními a přijímajícími stavy.

Systémy přepisovacích pravidel

Systémy přepisovacích pravidel mají identifikaci „RS“. Formát jejich uložení je velmi prostý. Nejdříve je na samostatném řádku neredkuvatelný řetězec systému a na dalších řídcích pak vždy dvojice levé a pravé strany přepisovacího pravidla oddělená tabulátorem:
RS
$< neredukovateln\_etzec >$
seznam pravidel ve tvaru:
$< lev\_strana >< tab >< prav\_strana >$



Obrázek B.2: Příklad uložení systému přepisovacích pravidel s neredukovatelným řetězcem „“ a pravidly $\$aa \rightarrow \$a$ a $\$ab\& \rightarrow \$\&$