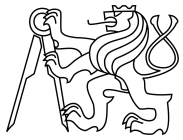
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Minimalizace stromových a zásobníkových automatů
Student:	Bc. Štěpán Plachý
Vedoucí:	Ing. Jan Trávníček
Studijní program:	Informatika
Studijní obor:	Systémové programování
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Nastudujte teorii stromových automatů, především minimalizaci deterministických stromových automatů. Nastudujte možnosti převodu stromových automatů na zásobníkové automaty.
Adaptujte algoritmus minimalizace deterministických stromových automatů na deterministické zásobníkové automaty vyšlé z převodu deterministických stromových automatů na zásobníkové.
Implementujte algoritmus minimalizace deterministických stromových automatů, algoritmus převodu stromových automatů na zásobníkové automaty a Vámi navržený algoritmus minimalizace deterministických zásobníkových automatů.
Otestujte implementované algoritmy, diskutujte vlastnosti algoritmu minimalizace deterministických zásobníkových automatů a možnosti dalšího zobecnění na větší třídu deterministických zásobníkových automatů.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 30. ledna 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Teoretické Informatiky

Diplomová práce

Minimalizace stromových a zásobníkových automatů

Bc. Štěpán Plachý

Vedoucí práce: Ing. Jan Trávníček

9. května 2017

Poděkování

Děkuji Ing. Janu Trávníčkovi za vedení této práce. Dále děkuji své rodině za velkou podporu při studiu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 9. května 2017 .....

Odkaz na tuto práci

Plachý, Štěpán. *Minimalizace stromových a zásobníkových automatů.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Práce se zabývá vztahem mezi konečnými stromovými automaty a zásobníkovými automaty, a jejich minimalizací. Konečné Stromové automaty jsou rozšířením konečných automatů a zpracovávají regulární stromové jazyky. V práci jsou popsané algoritmy redukce a minimalizace konečných stromových automatů, jejich převodu na zásobníkové automaty přijímající ekvivalentní jazyk v postfixovém zápisu a minimalizací převedených automatů. Všechny algoritmy jsou dále implementovány do projektu Automatová knihovna.

Klíčová slova minimalizace, stromový jazyk, konečný stromový automat, zásobníkový automat, převod stromového a zásobníkového automatu

Abstract

The topic of this thesis is the relationship between finite tree automata and pushdown automata and their minimization. Finite tree automata are an extension of finite automata and they process regular tree languages. In the thesis are described algorithms for reduction and minimization of finite tree automata, their conversion to pushdown automata accepting equivalent language in postfix notation and minimization of converting automata. All algorithms are also implemented in the project Automata library.

Keywords minimization, tree language, finite tree automaton, pushdown automaton, finite tree automaton and pushdown automaton conversion

Obsah

Seznam obrázků

Úvod

Automaty jsou formalismem pro zpracování jazyků. Daný jazyk ale může přijímat více různých automatů s rozdílnou velikostí reprezentace. Problém minimalizace je nalezení nejmenšího automatu přijímajícího totožný jazyk.

Konečný stromový automat je rozšíření konečného automatu pro zpracování regulárních stromových jazyků. Podobně, jako pro konečné automaty, ke každému stromovému automatu existuje unikátní minimální automat. Prvním krokem pro minimalizaci je redukce, neboli odstranění nedosažitelných a zbytečných stavů. Samotný algoritmus minimalizace poté hledá stavy s ekvivalentním chování, čímž se sníží počet stavů a tím i velikost celé reprezentace automatu. Díky unikátnosti minimálního automatu je také možné rozhodnout o ekvivalenci automatů, tedy zda přijímají totožný jazyk.

Stromy lze linearizací převést na řetězce. Z regulárního stromového jazyku tím vnikne bezkontextový jazyk, který je přijímám zásobníkovým automatem. Zásobníkové automaty mají na linearizovaných stromech větší výpočetní sílu, než stromové automaty. Stromový automat lze proto převést na ekvivalentní zásobníkový automat.

Automatová knihovna je projekt pro práci s formalismy z teorie automatů. V bakalářské práci [1] byla rozšířena knihovna o struktury stromových automatů a jejich determinizace. Cílem této práce je na základě existujících struktur navrhnout a implementovat algoritmy převodu stromových automatů na zásobníkové a minimalizace jak stromových, tak převedených zásobníkových automatů. Dalším cílem je diskutovat o možném zobecnění vytvořeného algoritmu minimalizace na větší třídu zásobníkových automatů.

Kapitola 1

Teorie

1.1 Abeceda

Definice 1.1. *Abeceda* Σ je neprázdná konečná množina. Prvek této množiny nazýváme *symbol*.

Definice 1.2. *Ohodnocený symbol* s_r je dvojice (s, r), kde s je symbol a $r \in N$ nazýváme aritou symbolu.

Definice 1.3. Buď s_r ohodnocený symbol s aritou r, pak funkce $arity(s_r) = r$.

Příkladem ohodnocených symbolů jsou funkce. Název funkce je symbol a arita je počet parametrů.

Definice 1.4. *Ohodnocená abeceda* F je neprázdná konečná množina ohodnocených symbolů.

Ohodnocená abeceda může obsahovat jeden symbol s více různými aritami. V takovém případě se jedná o různé ohodnocené symboly.

Příklad 1.1. Mějme abecedu $F = \{and_2, or_2, not_1, 1_0, 0_0\}$.

F je abecedou booleovských výrazů. Symboly s aritou 0 jsou konstanty a ostatní symboly jsou operátory. Arita operátorů označuje počet jejich operandů.

1.2 Řetězec

Definice 1.5. Buď Σ abeceda. Konečná posloupnost symbolů ze Σ se nazývá *Řetězec* nad abecedou Σ.

Definice 1.6. ϵ značí prázdný řetězec.

Definice 1.7. Buď $s = s_1 \dots s_i \dots s_n$ řetězec nad abecedou Σ, kde $i, n \in N, i \leq n$. Řetězec $s_1 \dots s_i$ se nazývá *prefix* řetězce s. Řetězec $s_i \dots s_n$ se nazývá *suffix* řetězce s.

1.3 Formální jazyk

Definice 1.8. Buď Σ abeceda. Σ^n označuje množinu všech řetězců nad Σ délky $n \in N$. Σ^* označuje množinu všech řetězců nad Σ.

Protože délka řetězce není omezena, je množina všech řetězců nad abecedou nekonečná.

Definice 1.9. *Formální jazyk* L na abecedou Σ je množina řetězců nad Σ, $L \subseteq \Sigma^*$.

Příklad 1.2. Mějme abecedu $\Sigma = \{0, 1\}$ a jazyk $L = \{\epsilon, 0, 1, 00, 01, 10, 11\}$ nad Σ.

L je jazykem všech řetězců nad Σ délky nejvýše 2.

1.4 Strom

Definice 1.10. *Strom* nad abecedou Σ je nelineární struktura vrcholů t definovaná rekurzivním vztahem

$$t = (s, c)$$
$$c = \{t_1 \dots t_n\}$$

kde $s \in \Sigma, n \in \mathbb{N}$.

Definice 1.11. Funkce $value(t) = c$.

Definice 1.12. Strom je *uspořádaný*, pokud c je uspořádaná n-tice.

Definice 1.13. *Potomek* vrcholu t je každý vrchol $t_i \in c$

Definice 1.14. *Rodič* vrcholu t je vrchol t_r, kde $t \in c_r$

Definice 1.15. *Sourozenec* vrcholu je každý vrchol, který má stejného rodiče.

Definice 1.16. *Kořen* je vrchol, který nemá rodiče.

Definice 1.17. *Stupeň* vrcholu t je $deg(t) = |c|$

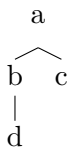
Definice 1.18. *List* je každý vrchol t, kde $deg(t) = 0$

Definice 1.19. *Vnitřní vrchol* je každý vrchol t, kde $deg(t) \neq 0$

Definice 1.20. *Hloubka* vrcholu je vzdálenost vrcholu[1] od kořenu.

Definice 1.21. *Výška* stromu je největší hloubka listů.

V celé této práci se uvažují pouze uspořádané stromy.



Obrázek 1.1: Příklad stromu

Příklad 1.3. Mějme strom na obrázku 1.1 nad abecedou $\Sigma = \{a, b, c, d\}$.

Vrchol a je kořenem, vrcholy a a b jsou vnitřní vrcholy a vrcholy c a d jsou listy stromu.

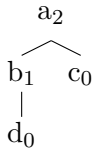
Vrcholy b a c jsou potomky vrcholu a a vrchol a je rodičem vrcholů b a c. Podobně vrchol d je potomkem vrcholu b a vrchol b je rodičem vrchol d. Vrcholy b a c jsou sourozenci.

Stupeň vrcholu a je 2, vrcholu b 1, a vrcholů c a d 0.

Hloubka vrcholu a je 0, vrcholů b a c 1 a vrcholu d 2. Výška stromu je 2.

Definice 1.22. Strom nad ohodnocenou abecedou F je *ohodnocený*, pokud pro každý vrchol t platí $deg(t) = arity\,(value(t))$

Příklad 1.4. Mějme strom na obrázku 1.2 nad abecedou $F = \{a_2, b_1, c_0, d_0\}$.



Obrázek 1.2: Příklad ohodnoceného stromu

Stupeň každého vrcholu je arita jeho symbolu.

1.5 Formální stromový jazyk

Definice 1.23. Buď Σ abeceda. T_Σ označuje množinu všech stromů nad abecedou Σ.

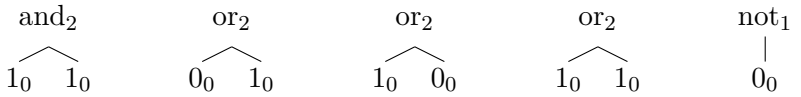
Definice 1.24. Buď F ohodnocená abeceda. TR_F označuje množinu všech ohodnocených stromů nad abecedou F.

Definice 1.25. *Formální stromový jazyk* L nad abecedou Σ je množina stromů nad Σ, $L \subseteq T_\Sigma$.

[1]Vzdáleností je myšleno počet vrcholů, které je třeba projít cestou od jednoho vrcholu k druhému.

Definice 1.26. *Formální ohodnocený stromový jazyk* L nad ohodnocenou abecedou F je množina ohodnocených stromů nad F, $L \subseteq TR_F$.

Příklad 1.5. Mějme ohodnocený stromový jazyk L nad abecedou $F = \{and_2, or_2, not_1, 1_0, 0_0\}$tvořený množinou stromů na obrázku 1.3.



Obrázek 1.3: Příklad ohodnoceného stromového jazyku

L je jazykem syntaktických stromů pravdivých booleovských výrazů s hloubkou 1.

1.6 Linearizace stromu

Definice 1.27. *Linearizace stromu* je zobrazení $T_\Sigma \to \Sigma'^*$, kde Σ a Σ' jsou abecedy.

Důsledek 1.27.1. Pro stromový jazyk L a linearizaci $f(t)$ jsou jazyky L a $f(L)$ ekvivalentní, pokud $f(t)$ je bijekce.

Definice 1.28. *Prefixový zápis* stromu t nad abecedou Σ je linearizace $prefix(t) \in T_\Sigma \to \Sigma^*$ s předpisem

$$prefix(t) = value(t)\, prefix\,(t[1]) \ldots prefix\,(t[deg(t)])$$

Definice 1.29. *Postfixový zápis* stromu t nad abecedou Σ je linearizace $postfix(t) \in T_\Sigma \to \Sigma^*$ s předpisem

$$postfix(t) = postfix\,(t[1]) \ldots postfix\,(t[deg(t)])\ \ value(t)$$

Příklad 1.6. Mějme strom na obrázku 1.1. Prefixový zápis stromu je *abdc* a postfixový je *dbca*.

Obrázek 1.4: Příklad stromů se stejným prefixovým zápisem

Prefixový i postfixový zápis nejsou pro obecný stromový jazyk bijekcí. Například oba stromy na obrázku 1.4 se prefixovým zápisem převedou na *abdc*.

Z prefixového i postfixového zápisu nelze zpětně určit stromovou strukturu, protože neobsahuje informaci o počtu potomků daného vrcholu. Oba zmíněné stromy mají vrcholy b a d, ale v obou případech mají jiné stupně.

U ohodnocených stromů je jejich struktura jednoznačně určena symboly ve vrcholech, proto řetězec pro každý ohodnocený strom v prefixovém i postfixovém zápisu je unikátní a obě linearizace jsou pro ohodnocené stromy bijekcí.

1.7 Stromový automat

Stromový automat je abstraktní výpočetní model pro rozhodnutí, zda strom patří do stromového jazyka.

Stromové automaty se rozdělují na několik druhů. Podle směru vyhodnocování se dělí na Bottom-up a Top-down. Dále se rozlišuje, za zpracovávají ohodnocený nebo neohodnocený stromový jazyk.

Automaty zpracovávající neohodnocené stromové jazyky jsou složitějším modelem, protože nemají omezenou přechodovou funkci. V této práci se uvažují pouze automaty pro ohodnocené stromové jazyky.

1.7.1 Bottom-up stromový automat

Definice 1.30. *BU NFTA* je čtveřice $A = (Q, F, Q_f, \Delta)$, kde Q je množina stavů, F je ohodnocená abeceda, $Q_f \subseteq Q$ je množina finálních stavů a Δ je zobrazení $F \times Q^* \to 2^Q$.

Definice 1.31. Buď $A = (Q, F, Q_f, \Delta)$ BU NFTA, t ohodnocený strom nad abecedou F a $\Delta_t(t)$ funkce definovaná vztahem:

$$\Delta_t(t) = \Delta\left(value(t), \{\Delta_t(t[1]) \dots \Delta_t(t[n])\}\right)[i]$$

pro libovolné $i \in \mathbb{N}$, kde $n = arity(value(t))$.

Říkáme, že BU NFTA A *přijímá* ohodnocený strom t, pokud $\exists \Delta_t(t) \in Q_f$, v opačném případě říkáme, že automat ohodnocený strom *nepřijímá*.

Přechodová funkce danému vrcholu přiřazuje stav z jeho symbolu a stavu jeho potomků. Pro její výpočet se nejprve rekurzivně vyhodnotí stav potomků. Zapisovat se bude výčtem mapovacích pravidel ve tvaru

$$f(q_1 \dots q_n) \to q$$

pro symbol $f \in F$, stav i-tého potomka $q_i \in Q$, $n = arity(f)$, a výsledný stav $q \in Q$.

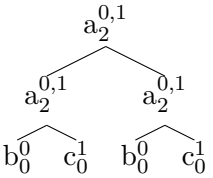
Nedeterminismus zde znamená, že může existovat více pravidel, které mají stejnou levou stranu, ale rozdílnou pravou stranu.

Příklad 1.7. Mějme jazyk všech binárních stromů s vnitřními vrcholy a a listy b a c, kde b je vždy levým potomkem a c vždy pravým. Vytvořme BU NFTA přijímající tento jazyk.

$A = (Q, F, Q_f, \Delta)$
$F = \{a_2, b_0, c_0\}$
$Q = \{0, 1\}$
$Q_f = \{0, 1\}$
Δ :

$$\begin{aligned} b &\to 0 \\ c &\to 1 \\ a(0,1) &\to 0 \\ a(0,1) &\to 1 \end{aligned}$$

Stav 0 označuje, že podstrom je levým potomkem, 1 pravým. Vrcholy b a c mohou být pouze levým resp. pravým potomkem, kdežto vrchol a může být obojí. Kvůli tomu má pravidlo $a(0,1)$ více pravých stran. To způsobuje nedeterminismus při vyhodnocování. Ukažme si výsledek průchodu na následujícím stromu:



Obrázek 1.5: Výsledek příkladu 1.7

V horním indexu symbolu jsou všechny stavy, ve kterých se může vrchol nacházet. Jelikož alespoň jeden ze stavů kořene je z množiny finálních stavů, automat strom přijímá.

1.7.1.1 Deterministický bottom-up stromový automat

Definice 1.32. *BU DFTA* je čtveřice $A = (Q, F, Q_f, \Delta)$, kde Q je množina stavů, F je množina vstupních ohodnocených symbolů, $Q_f \subseteq Q$ je množina finálních stavů a Δ je zobrazení $F \times Q^* \to Q$.

Příklad 1.8. Vytvořme deterministickou variantu automatu z příkladu 1.7.

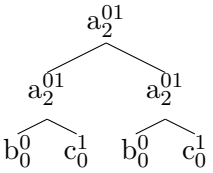
$A = (Q, F, Q_f, \Delta)$
$F = \{a_2, b_0, c_0\}$
$Q = \{0, 1, 01\}$
$Q_f = \{0, 1, 01\}$

Δ :

$$\begin{aligned} b &\to 0 \\ c &\to 1 \\ a(0,1) &\to 01 \\ a(01,1) &\to 01 \\ a(0,01) &\to 01 \\ a(01,01) &\to 01 \end{aligned}$$

Jelikož pravidlo pro symbol a způsobovalo nedeterminismus, jelikož vrchol se symbolem a může být na levé i pravé straně, přidáme třetí stav 01 označující, že vrchol může být vlevo i vpravo. Tím musíme přidat další pravidla pro každou konfiguraci vrcholu, pro kterou se mohou někteří potomci v tomto stavu nacházet.

Výsledek průchodu je v tomto případě následující:



Obrázek 1.6: Výsledek příkladu 1.8

Jelikož je automat deterministický, každý vrchol stromu se již nachází pouze v jednom stavu. Protože stav kořene je z množiny finálních stavů, automat strom, stejně jako v předchozím příkladě, přijímá.

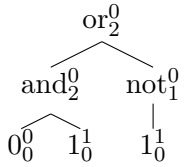
Příklad 1.9. Vytvořme automat přijímající všechny pravdivé konstantní logické výrazy.

$A = (Q, F, Q_f, \Delta)$
$F = \{and_2, or_2, not_1, 1_0, 0_0\}$
$Q = \{0, 1\}$
$Q_f = \{1\}$
Δ :

$$\begin{aligned} 0 &\to 0 & and(0,0) &\to 0 & or(0,0) &\to 0 \\ 1 &\to 1 & and(0,1) &\to 0 & or(0,1) &\to 1 \\ not(0) &\to 1 & and(1,0) &\to 0 & or(1,0) &\to 1 \\ not(1) &\to 0 & and(1,1) &\to 1 & or(1,1) &\to 1 \end{aligned}$$

Stavy označují pravdivostní ohodnocení vrcholu, pravidla proto vycházejí z tabulek pravdivostních ohodnocení logických operací. Výsledek průchodu si ukažme na stromu na obrázku 1.7.

Pravdivostní ohodnocení kořene je 0, což není koncový stav, proto automat tento strom nepřijímá.



Obrázek 1.7: Výsledek příkladu 1.9

1.7.2 Top-down stromový automat

Definice 1.33. *TD NFTA* je čtveřice $A = (Q, F, Q_s, \Delta)$, kde Q je množina stavů, F je množina vstupních ohodnocených symbolů, $Q_s \subseteq Q$ je množina počátečních stavů a Δ je zobrazení $F \times Q \to 2^{(Q^*)}$.

Definice 1.34. Buď $A = (Q, F, Q_s, \Delta)$ TD NFTA, t ohodnocený strom nad abecedou F a $\Delta_t(q, t)$ funkce definovaná vztahem:

$$\Delta_t(q,t) = (arity(value(t)) = 0) \vee (\Delta_{ti}(q,t,1) \wedge \cdots \wedge \Delta_{ti}(q,t,n))$$

$$\Delta_{ti}(q,t,i) = \Delta_t(\Delta(q, value(t))[i], t[i])$$

kde $q \in Q$ a $n = arity(value(t))$.

Říkáme, že TD NFTA A *přijímá* ohodnocený strom t, pokud $\exists\Delta_t(q_s, t)$ pro libovolné $q_s \in Q_s$, v opačném případě říkáme, že automat ohodnocený strom *nepřijímá*.

Mapovací pravidla pro TD automat budou mít tvar

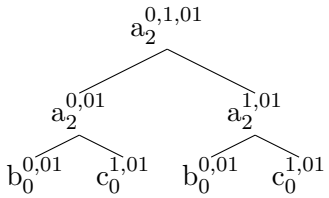
$$f(q) \to (q_1 \dots q_n)$$

pro stav vrcholu $q \in Q$, symbol vrcholu $f \in F$ a výsledný stav i-tého potomka $q_i \in Q$, $i \in \mathbb{N}$, $n = arity(f)$.

Příklad 1.10. Vytvořme TD verzi automatu z příkladu 1.8.
$A = (Q, F, Q_s, \Delta)$
$F = \{a_2, b_0, c_0\}$
$Q = \{0, 1, 01\}$
$Q_s = \{0, 1, 01\}$
Δ :

$$\begin{aligned} b(0) &\to () \\ c(1) &\to () \\ a(01) &\to (0,1) \\ a(01) &\to (01,1) \\ a(01) &\to (0,01) \\ a(01) &\to (01,01) \end{aligned}$$

Automat vznikl z původního BU automatu prohozením stavů na levé a pravé straně, čímž se otočí směr vyhodnocování. Ač byl původní automat deterministický, nový automat již není. Výsledek průchodu v tomto případě vypadá následovně:



Obrázek 1.8: Výsledek příkladu 1.10

V horním indexu se nacházejí stavy, které mohli být vrcholu přiřazeny z rodiče nebo z množiny počátečních stavů pro kořen. Jelikož každému listu byl přiřazen alespoň jeden stav, pro který existuje mapovací pravidlo, automat strom přijímá.

1.7.2.1 Deterministický top-down stromový automat

Definice 1.35. *TD DFTA* je čtveřice $A = (Q, F, q_s, \Delta)$, kde Q je množina stavů, F je množina vstupních ohodnocených symbolů, $q_s \in Q$ je počáteční stav a Δ je zobrazení $F \times Q \to Q^*$.

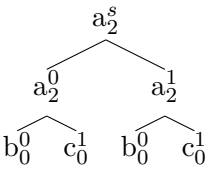
Pro přechodová pravidla platí, stejně jako u BU DFTA, že pro každou levou stranu může existovat pouze jedna pravá strana.

Příklad 1.11. Vytvořme deterministickou variantu automatu z příkladu 1.10.
$A = (Q, F, q_s, \Delta)$
$F = \{a_2, b_0, c_0\}$
$Q = \{0, 1, s\}$
$Q_s = s$
Δ :

$$b(s) \to ()$$
$$b(0) \to ()$$
$$c(s) \to ()$$
$$c(1) \to ()$$
$$a(s) \to (0, 1)$$
$$a(0) \to (0, 1)$$
$$a(1) \to (0, 1)$$

Tento automat vychází z nedeterministické verze BU automatu, jelikož si vystačíme s informací, zda je vrchol levým či pravým potomkem, protože to

můžeme z rodiče jednoznačně určit. Jelikož ale měl BU automat více finálních stavů, přidáme další stav s (a k němu odpovídající pravidla), který bude počáteční a bude značit, že vrchol je kořen. Výsledek průchodu v tomto případě vypadá následovně:



Obrázek 1.9: Výsledek příkladu 1.11

Jelikož je automat deterministický, stejně jako v případě BU DFTA je každému vrcholu přiřazen pouze jeden stav a protože každý list má přiřazený stav, pro který existuje mapovací pravidlo, automat strom přijímá.

1.7.3 Převod mezi Top-down a Bottom-up stromovým automatem

Ke každému TD automatu existuje ekvivalentní BU automat a opačně, mají proto stejnou výpočetní sílu. BU automat lze navíc determinizovat, proto deterministická varianta je také stejně výpočetně silná. To neplatí pro TD automat a jeho deterministická varianta je tedy méně výpočetně silná.[2]

Definice 1.36. Buď $A = (Q, F, Q_s, \Delta)$ TD NFTA, pak BU NFTA přijímající jazyk $L(A)$ je $A' = (Q, F, Q_f, \Delta')$, kde $Q_f = Q_s$ a $\Delta' = \{\Delta'(f, q_1 \dots q_n) = q : \Delta(f, q) = (q_1 \dots q_n)\}$

Důsledek 1.36.1. BU automat převedený z TD NFTA $A = (Q, F, Q_s, \Delta)$ je deterministický, pokud $\forall q, q' \in Q, q \neq q', \forall f \in F : \Delta(f, q) = (q_1 \dots q_n) \implies \Delta(f, q') \neq (q_1 \dots q_n)$.

Důsledek 1.36.2. TD automat převedený z BU NFTA $A = (Q, F, Q_f, \Delta)$ je deterministický, pokud $|Q_f| = 1$ a $\forall (q_1 \dots q_n), (q'_1 \dots q'_n) \in Q^*, (q_1 \dots q_n) \neq (q'_1 \dots q'_n), \forall f \in F : \Delta(f, q_1 \dots q_n) = q \implies \Delta(f, q'_1 \dots q'_n) \neq q$.

Při převodu se otáčí směr vyhodnocování prohozením stavů z levé a pravé strany přechodových pravidel. Počáteční stavy se stávají koncové a opačně.

Příkladem konverze jsou automaty v příkladech 1.8 a 1.10. TD automat splňuje podmínky z důsledku 1.36.1 a BU automat je deterministický. V opačném případě ale BU automat nesplňuje podmínky z důsledku 1.36.2 a TD automat deterministický není.

V této práci se budou používat pouze BU stromové automaty, proto každý stromový automat bude automaticky BU.

1.8 Zásobníkový automat

Zásobníkový automat je výpočetní model pro rozhodnutí, zda řetězec patří do jazyka. Výpočetní silou se nachází mezi konečným automatem a lineárně omezeným turingovým strojem.

Na linearizovaných stromech je zásobníkový automat výpočetně silnější, než stromový automat.[2]

Definice 1.37. *NPDA* je sedmice $A = (Q, \Sigma, G, \delta, q_0, Z_0, Q_f)$, kde Q je množina stavů, Σ je abeceda, G je zásobníková abeceda, δ je zobrazení $Q \times (\Sigma \cup \{\epsilon\}) \times G^* \to 2^{Q \times G^*}$, q_0 je počáteční stav, Z_0 je počáteční zásobníkový symbol a Q_f je množina finálních stavů.

Definice 1.38. *Konfigurace NPDA* $A = (Q, \Sigma, G, \delta, q_0, Z_0, Q_f)$ je trojice $R = (q, w, \alpha)$, kde $q \in Q$ je vnitřní stav, $w \in \Sigma^*$ je suffix vstupního řetězce a $\alpha \in G^*$ je obsah zásobníku. *Počáteční konfigurace NPDA* pro vstupní řetězec a je $R_0 = (q_0, a, Z_0)$.

Definice 1.39. *Přechod NPDA* $A = (Q, \Sigma, G, \delta, q_0, Z_0, Q_f)$ je relace $(q, aw, \beta\alpha) \vdash_A (p, w, \beta\gamma) \subset (Q \times \Sigma^* \times G^*) \times (Q \times \Sigma^* \times G^*)$, kde $(p, \gamma) \in \delta(q, a, \alpha)$. Tranzitivně reflexivní uzávěr relace $\vdash_A$ je značen $\vdash_A^*$.

Zásobníkový automat přijímá jazyk buď finálním stavem nebo prázdným zásobníkem. Pokud přijímá prázdným zásobníkem, je množina finálních stavů prázdná.

Definice 1.40. NPDA $A = (Q, \Sigma, G, \delta, q_0, Z_0, Q_f)$ řetězec $a \in \Sigma^*$ *přijímá finálním stavem*, pokud $(q_0, a, Z_0) \vdash_A^* (q, \epsilon, \gamma)$, kde $\gamma \in G^*$ a $q \in Q_f$.

Definice 1.41. NPDA $A = (Q, \Sigma, G, \delta, q_0, Z_0, \emptyset)$ řetězec $a \in \Sigma^*$ *přijímá prázdným zásobníkem*, pokud $(q_0, a, Z_0) \vdash_A^* (q, \epsilon, \epsilon)$, kde $q \in Q$.

1.8.1 Deterministický zásobníkový automat

Definice 1.42. Zásobníkový automat $A = (Q, \Sigma, G, \delta, q_0, Z_0, Q_f)$ je *deterministický*, pokud:

1. $\forall q \in Q, \forall a \in \Sigma, \forall \alpha \in G^* : |\delta(q, a, \alpha)| \leq 1$
2. Pokud $\delta(q, a, \alpha) \neq \emptyset, \delta(q, a, \beta) \neq \emptyset, \alpha \neq \beta$, pak α není prefix β a β není prefix α.
3. Pokud $\delta(q, a, \alpha) \neq \emptyset, \delta(q, \epsilon, \beta) \neq \emptyset, \alpha \neq \beta$, pak α není prefix β a β není prefix α.

V práci bude použita následující pomocná funkce:

Definice 1.43. Buď $A = (Q, \Sigma, G, \delta, q, Z_0, Q_F)$ DPDA. Funkce δ_G je zobrazení $Q \times \Sigma \times G^* \to G$, pro které platí $\delta_G(q, a, \alpha) = \beta : \delta(q, a, \alpha) \to (q', \beta) \in \delta$.

Kapitola 2

Automatová knihovna

Automatová knihovna je projekt zaměřený na práci s formalizmy z teorie automatů. Je vyvíjena na katedře teoretické informatiky ČVUT a je psána v jazyce C++. Tato práce je pokračováním z bakalářské práce [1], kde byly do knihovny přidány struktury stromových automatů a stromů, základní algoritmy nad nimi a determinizace stromových automatů.

Základem projektu jsou dvě dynamické knihovny `libalib2data.so`, sdružující datové struktury, a `libalib2algo.so`, sdružující algoritmy nad těmito strukturami.

Nad dynamickými knihovnami je pak tvořena sada jednoúčelových programů řešící konkrétní problém. Tyto programy lze spojovat unixovými roury pro dosažení složitějšího výpočtu. Datové struktury na vstupu a výstupu jsou ve formátu XML.

V této práci je přidána funkcionalita pro programy `atrim2`, `aminimize2` a `aconversion2`

2.1 Program atrim2

Program `atrim2` slouží k redukci automatů a gramatik. V případě automatů odstraňuje nedosažitelné a zbytečné stavy.

Odstranění nedosažitelných stavů se nastavuje přepínačem `-r` a zbytečných stavů přepínačem `-u`. Lze použít oba přepínače zároveň.

Program čte datovou strukturu ve formátu XML ze standardního vstupu, případně ze souboru při použití přepínače `-i`. Na výstupu je redukovaná vstupní datová struktura ve formátu XML, případně chyba, pokud redukce struktury není podporována.

2.2 Program aminimize2

Program `aminimize2` slouží k minimalizaci libovolného redukovaného deterministického automatu na vstupu. Algoritmus minimalizace je zvolen podle datové struktury na vstupu. U konečných automatů je implementováno více algoritmů a dají se zvolit.

Program čte deterministický automat ve formátu XML ze standardního vstupu, případně ze souboru při použití přepínače `-i`. Výstupem je minimalizovaný automat ve formátu XML, případně chyba, pokud minimalizace struktury na vstupu není podporována.

2.3 Program aconversion2

Program `aconversion2` převádí datové struktury mezi různými formalismy. V této práci se hlavně jedná o převod stromových automatů na zásobníkové.

Cílový formalismus se volí přepínačem `-t <název>`. Pro zásobníkový automat je název `pda`. Pro některé převody je implementováno více algoritmů a lze mezi nimi volit.

Program čte datovou strukturu ve formátu XML na vstupu, případně ze souboru při použití přepínače `-i`. Na výstupu je zvolená cílová datová struktura ve formátu XML, případně chyba, pokud převod není podporován.

Kapitola 3

Redukce stromového automatu

Automat může obsahovat stavy, do kterých se v průchodu nad libovolným vstupem nikdy nedostane, nebo ze kterých nikdy nedojde do koncového stavu. Těmto stavům se říká nedosažitelné a zbytečné.

Důvody k jejich odstranění jsou nejen pro zmenšení automatu, ale některé algoritmy kvůli nim nemusí pracovat správně, například minimalizace.

Definice 3.1. Buď $A = (Q, F, Q_f, \Delta)$ NFTA. Stav q je *dosažitelný* pokud platí alespoň jedna z podmínek:

1. $\exists f \in F : \Delta(f, \emptyset) = q$
2. $\exists f \in F : \Delta(f, q_1 \dots q_n) = q$, kde $q_1 \dots q_n$ jsou dosažitelné stavy.

V opačném případě je stav *nedosažitelný*.

Definice 3.2. Buď $A = (Q, F, Q_f, \Delta)$ NFTA. Stav q je *užitečný* pokud platí alespoň jedna z podmínek:

1. $q \in Q_f$
2. $\exists f \in F : \Delta(f, q_1 \dots q \dots q_n) = q'$, kde q' je užitečný stav.

V opačném případě je stav *zbytečný*.

Odstraněním zbytečných a nedosažitelných stavů se také zmenší přechodová funkce, a to o každé pravidlo, které obsahovalo jeden z těchto stavů.

3.1 Návrh algoritmu

Pro redukci se z automatu odstraní nedosažitelné stavy a následně zbytečné stavy. Algoritmus má proto dvě funkce, *removeUselessStates* a *removeUnreachableStates*, popsané v následujících sekcích.

Algoritmus 1 Redukce NFTA

Vstup: NFTA $A = (Q, F, Q_f, \Delta)$
Výstup: NFTA $A' = (Q', F, Q'_f, \Delta')$
return $removeUselessStates\,(removeUnreachableStates\,(A))$

3.1.1 Odstranění zbytečných stavů

Finální stavy jsou podle definice 3.2 implicitně užitečné. Užitečné jsou dále všechny stavy na levé straně přechodového pravidla, které má na pravé straně užitečný stav.

Princip algoritmu je tedy vkládat užitečné stavy do fronty a při zpracování prvku z fronty se vyhledají další užitečné stavy. U každého stavu ve frontě se naleznou všechna přechodová pravidla, která mají na pravé straně daný stav a všechny stavy na levé straně, které dosud nebyli nalezeny, se označí užitečnými a vloží do fronty. Každé takové pravidlo se také bude nacházet ve výsledném automatu, protože všechny jeho stavy jsou užitečné. Algoritmus končí, když je fronta prázdná.

Jazyk přijímaný automatem bez finálních stavů je prázdný. V takovém automatu jsou všechny stavy zbytečné a stejně tak i přechodová pravidla.

Algoritmus je popsán pseudokódem 2.

3.1.2 Odstranění nedosažitelných stavů

Odstranění nedosažitelných stavů je podobné odstranění zbytečných stavů, ale složitější. Podle definice 3.1 jsou implicitně dosažitelné stavy, které jsou na pravé straně pravidla s konstatním symbolem, ale všechny ostatní stavy jsou dosažitelné, pokud jsou na pravé straně pravidla, kde jsou dosažitelné všechny stavy na levé straně.

U každého přechodového pravidla se bude držet informace o počtu nedosažitelných stavů na levé straně, což jsou na začátku všechny. U každého stavu se bude držet informace, ve kterých pravidlech se nachází na levé straně a kolikrát.

Algoritmus podobně bude mít frontu dosažitelných stavů. Z každým stavem ve frontě se projdou všechna pravidla, kde se daný stav nachází na levé straně a sníží se u nich počet nedosažitelných stavů o počet výskytů daného stavu.

Pokud počet výskytů nedosažitelných stavů v pravidlu je nula, je stav na pravé straně dosažitelný a vloží do fronty. Pravidlo v takové situaci již obsahuje pouze dosažitelné stavy, a bude se nacházel ve výsledném automatu.

Algoritmus je popsán pseudokódem 3.

Algoritmus 2 Odstranění zbytečných stavů NFTA

Vstup: NFTA $A = (Q, F, Q_f, \Delta)$
Výstup: NFTA $A' = (Q', F, Q_f, \Delta')$

```
Q' ← Q_f
queue ← ∅
for all q_f ∈ Q_f do
  queue.enqueue(q_f)
end for
transitionsToState : Q → 2^Δ
for all δ = ((f, q_1 ... q_n) → q) ∈ Δ do
  transitionsToState ← transitionsToState ∪ {q → δ}
end for
while queue ≠ ∅ do
  q ← queue.dequeue()
  for all δ = ((f, q_1 ... q_n) → q) ∈ transitionsToState (q) do
    for i ← 1 to n do
      if q_i ∉ Q' then
        Q' ← Q' ∪ {q_i}
        queue.enqueue (q_i)
      end if
    end for
    Δ' ← Δ' ∪ {δ}
  end for
end while
return A'
```

3.2 Implementace

3.2.1 Odstranění zbytečných stavů

Odstranění zbytečných stavů je implementováno podle pseudokódu 2. Implementace se nachází v souboru `UselessStatesRemover.h` pro deterministické automaty.

Jako fronta stavů je použita standardní fronta `std::deque<StateType>`. Pro uchování pravidel obsahujících určitý stav je použita standardní mapa stavu a množiny ukazatelů na konstantní strukturu přechodových pravidel. `std::map<StateType, std::set<const Transition *>>`.

3.2.2 Odstraněný nedosažitelných stavů

Odstranění zbytečných stavů je implementováno podle pseudokódu 3. Implementace se nachází v souboru `UnreachableStatesRemover.h` pro deterministické automaty.

Algoritmus 3 Odstranění nedosažitelných stavů NFTA

Vstup: NFTA $A = (Q, F, Q_f, \Delta)$
Výstup: NFTA $A' = (Q', F, Q'_f, \Delta')$

```
queue ← ∅
stateOccurrences : Q → 2^(Δ×N)
unreachableCount ← ∅
for all δ = ((f, q_1 ... q_n) → q) ∈ Δ do
  if arity(f) = 0 then
    Q' ← Q' ∪ {q}
    queue.enqueue(q)
    Δ' ← Δ' ∪ {δ}
  else
    unreachableCount ← unreachableCount ∪ {(δ, arity(f))}
    for i = 0 to n do
      if (δ, c) ∈ stateOccurences (q_i) , c ∈ N then
        c ← c + 1
      else
        stateOccurrences (q_i) ← stateOccurrences (q_i) ∪ {(δ, 1)}
      end if
    end for
  end if
end for
while queue ≠ ∅ do
  q ← queue.dequeue()
  for all (δ = ((f, q_1 ... q_n) → q') , c) ∈ stateOccurrences (q) do
    unreachableCount (δ) ← unreachableCount (δ) − c
    if unreachableCount (δ) = 0 then
      Q' ← Q' ∪ {q'}
      queue.enqueue (q')
      Δ' ← Δ' ∪ {δ}
    end if
  end for
end while
Q'_f ← Q_f ∩ Q'
return A'
```

Pro uchování počtu nedosažitelných stavů v každém přechodovém pravidlu je použit standardní vektor páru odkazu na strukturu přechodových pravidel a celé číslo `std::vector<std::pair<const Transition *, int>>`. Pro uchování počtu výskytů konkrétního stavu v pravidlech je použita standardní mapa ukazatele na uvedený pár pravidla s počtem nedosažitelných stavů a celé číslo `std::map<std::pair<const Transition *, int> *, int>`. Uchování páru místo samotného pravidla ušetří dotazy na počet nedosažitelných stavů. Celá

struktura pro uchování počtu výskytů stavů v pravidlech je tedy mapa stavu a předchozí mapy:

```
std::map<StateType,
         std::map<std::pair<const Transition *, int> *, int>>
```

Kapitola 4

Minimalizace stromového automatu

Minimalizace automatu je problém nalezení nejmenší možné reprezentace automatu přijímající totožný jazyk. Jak u konečných, tak u stromových automatů je velikost reprezentace omezena velikostí množiny stavů. Problém proto v obou případech je limitován na zmenšení množiny stavů, která vyústí ve zmenšení přechodové funkce o pravidla obsahující vyřazené stavy.

Pokud je nějaký stav v automatu navíc, znamená to, že se pro libovolný vstup chová stejně, jako jiný stav.

V [2] je dokázáno, že ke stromovému automatu existuje unikátní minimální automat, pokud se nebere v úvahu pojmenování stavů. Zároveň popisuje princip algoritmu jeho nalezení.

Protože je minimální automat unikátní, lze minimalizaci použít pro rozhodnutí ekvivalence automatů porovnáním jejich minimálních reprezentací.

4.1 Návrh algoritmu

Algoritmus na vstupu vyžaduje redukovaný deterministický automat. Princip algoritmu je sdružení stavů do tříd ekvivalence a jejich následné dělení při nesplnění podmínek, dokud je takové dělení možné.

Implicitně nejsou ekvivalentní finální a nefinální stavy, proto tvoří počáteční dvě třídy ekvivalence. Dělení tříd ekvivalence probíhá podle následující definice:

Definice 4.1. Nechť $A = (Q, F, Q_f, \Delta)$ je redukovaný DFTA. Pro stavy q a q' a i-tou relaci ekvivalence P_i platí $qP_{i+1}q'$, pokud:

1. qP_iq'
2. $\Delta(f, q_1 \dots q_{j-1}, q, q_{j+1} \dots q_n)\, P_i \Delta(f, q_1 \dots q_{j-1}, q', q_{j+1} \dots q_n)$, $\forall f \in F, \forall q_1 \dots q_{j-1}, q_{j+1} \dots q_n \in Q$

Podle bodu 2. z definice, aby byli dva stavy nadále ve stejné třídě ekvivalence, musí mít všechny dvojice pravidel, jejichž levé strany se liší těmito dvěma stavy na jedné konkrétní pozici, mít na pravé straně stavy ze stejné třídy ekvivalence.

To zároveň implikuje, že pokud se v nějakém pravidle nachází jeden z těchto stavů vícekrát, musí existovat pravidlo pro každou variaci těchto dvou stavů na pozicích výskytu. Pokud tedy například o stavech 1 a 2 tvrdíme, že jsou ekvivalentní a v automatu existuje pravidlo s levou stranou:

$$f\,(0,1,0,1,0)$$

pak musí v automatu existovat pravidla s levými stranami:

$$f\,(0,1,0,1,0)$$
$$f\,(0,1,0,2,0)$$
$$f\,(0,2,0,1,0)$$
$$f\,(0,2,0,2,0)$$

Všechny stavy na pravých stranách těchto pravidel musí být zároveň vzájemně ekvivalentní.

Automat nemůže obsahovat nedosažitelné stavy, protože by mohly zapříčinit rozdělení ekvivalentních stavů do různých tříd ekvivalence. Pokud například máme nedosažitelný stav n, dva ekvivalentní stavy 0 a 1, dva neekvivalentní stavy 3 a 4, a následující pravidla:

$$f\,(n,0) \to 3$$
$$f\,(n,1) \to 4$$

pak by stavy 0 a 1 nesplňovaly podmínky z definice. Tato pravidla ale nikdy nebudou v automatu použita, proto ekvivalenci stavů nijak neovlivňují. Odstraněním nedosažitelných stavů by se odstranila i tato pravidla a k této situaci poté nedojde.

Na rozdíl od nedosažitelných stavů, zbytečné stavy automat obsahovat může, ale za předpokladu, že je automat úplný, tedy že v něm existuje pravidlo pro každou levou stranu. Pokud je při vyhodnocení použit libovolný zbytečný stav, strom nebude přijat. Mají proto stejné chování pro libovolný vstup a jsou tedy vzájemně ekvivalentní. Pokud by se vytvořil úplný automat doplněním chybějících pravidel a na jejich pravou stranu dal nefinální stav q_{fail}, každý zbytečný stav by s ním byl ekvivalentní. Pokud se na levé straně pravidla nachází zbytečný stav, je i na pravé straně zbytečný stav. Pokud tedy například máme zbytečné stavy z_1, z_2 a z_3, dva ekvivalentní stavy 0 a 1, a pravidla:

$$f\,(z_1,0) \to z_2$$
$$f\,(z_1,1) \to z_3$$

pak stavy 0 a 1 splňují definici, protože zbytečné stavy na pravé straně jsou ekvivalentní. Pokud ale automat není úplný, jsou obě pravidla zbytečné a v automatu nemusí být, a pokud jedno pravidlo automat obsahuje a druhé ne, stavy 0 a 1 nesplňují definici, přestože jsou ekvivalentní.

Každé pravidlo $f(q_1 \dots q_{i-1} q_i q_{i+1} \dots q_n) \to q'$ lze pro libovolný index i zapsat do pětice:

$$(q_i, f, (q_1 \dots q_{i-1} q_{i+i} \dots q_n), q', i) \tag{4.1}$$

Pro každé pravidlo existuje n takových pětic, ale v opačném směru je pravidlo jednoznačné. Aplikováním definice na tento tvar tedy dostáváme, že stavy q a p patří v následující iteraci do stejné třídy ekvivalence, pokud:

$$(eq_j(q), f, (q_1 \dots q_{n-1}), eq_j(q'), i) = (eq_j(p), f, (q_1 \dots q_{n-1}), eq_j(p'), i)$$

pro všechna $i \leq n$ a $q_1 \dots q_n \in Q$, kde $eq_j(q)$ je třída ekvivalence stavu q v j-té iteraci a $(q, f, (q_1 \dots q_{n-1}), q', i), (p, f, (q_1 \dots q_{n-1}), p', i) \in \Delta$.

Pro každý stav q je tedy jeho třída ekvivalence v další iteraci jednoznačně určena množinou:

$$\{(eq_j(q), f, (q_1 \dots q_{n-1}), eq_j(q'), i) : (q, f, (q_1 \dots q_{n-1}), q', i) \in \Delta\} \tag{4.2}$$

Jelikož první člen pětice se nachází v každém členu množiny a druhý, třetí a čtvrtý člen jsou stejné více různých indexů, lze množinu zredukovat na strukturu:

$$(eq_j(q), \{((f, (q_1 \dots q_{n-1}), eq_j(q')), \{i_1 \dots i_k\})\}) \tag{4.3}$$

Princip navrhovaného algoritmu je v každé iteraci tuto strukturu pro každý stav zkonstruovat a na základě rovnosti těchto struktur spojit stavy do jedné třídy ekvivalence.

Příklad 4.1. Uvažujme v dané iteraci algoritmu stavy q_0, q_1, q_2 a q_3, dvě třídy ekvivalence $E_1 = \{q_0, q_1\}$ a $E_2 = \{q_3, q_4\}$, a přechodová pravidla:

$$\begin{aligned} f(q_0) &\to q_3 \\ f(q_1) &\to q_4 \\ g(q_0, q_0) &\to q_3 \\ g(q_0, q_1) &\to q_4 \\ g(q_1, q_0) &\to q_4 \\ g(q_1, q_1) &\to q_3 \end{aligned}$$

Stavy q_0 a q_1 podle zadaných pravidel splňují podmínky z definice 4.1, a měly by proto patřit do stejné třídy ekvivalence i v další iteraci. Struktura podle 4.3 charakterizující třídu ekvivalence v další iteraci pro oba stavy vypadá následovně:

$$\left(E_1, \left\{\begin{array}{c} ((f, \{\}, E_2), \{1\}) \\ ((g, \{q_0\}, E_2), \{1, 2\}) \\ ((g, \{q_1\}, E_2), \{1, 2\}) \end{array}\right\}\right)$$

Pro stav q_0 je první řádek struktury vytvořen z pravidla $f(q_0) \to q_3$, druhý řádek z pravidla $g(q_0, q_0) \to q_3$ a třetí řádek z pravidel $g(q_0, q_1) \to q_4$ a $g(q_1, q_0) \to q_4$.

Pro stav q_1 je první řádek struktury vytvořen z pravidla $f(q_1) \to q_4$, druhý řádek z pravidel $g(q_0, q_1) \to q_4$ a $g(q_1, q_0) \to q_4$, a třetí řádek z pravidla $g(q_0, q_0) \to q_3$.

4.2 Implementace

Algoritmus je implementován podle pseudokódu 4. Implementace se nachází v souboru `Minimize.h`.

Třídy ekvivalence se v každé iteraci tvoří pomocí mapy, kde klíč je celá struktura z 4.3 a hodnota je množina stavů. Po konstrukci struktury lze jedním příkazem vložit stav do správné množiny, protože porovnání je delegováno do celé struktury.

Místo ukládání tříd ekvivalencí do struktury se ukládá stav, který třídu reprezentuje, což je libovolný stav, který do dané třídy patří. V implementaci je to první stav z množiny.

Klíč je tvořen párem stavu reprezentujícího třídu ekvivalence a mapou s informacemi přechodových pravidlech, ve kterých se stav nachází. To je konkrétně trojice symbolu, vektoru stavů na levé straně pravidla a stavu, který reprezentuje třídu ekvivalence stavu na pravé straně pravidla. Druhým parametrem mapy je množina indexů, na kterých se stav v pravidle nachází.

Pro trojici je použita v knihovně implementovaná třída `tuple`, která je analogická k páru, ale pro tři prvky.

Výsledná struktura je tedy:

```
std::map<std::pair<StateType,
                   std::map<std::tuple<std::ranked_symbol<>,
                                       std::vector<StateType>,
                                       StateType>,
                            std::set<int>>>,
         std::set<StateType> >
```

Pro mapování stavu na stav reprezentující třídu ekvivalence je použita `std::map <StateType, StateType>`.

Algoritmus 4 Minimalizace redukovaného DFTA

Vstup: DFTA $A = (Q, F, Q_f, \Delta)$
Výstup: DFTA $A' = (Q', F, Q'_f, \Delta')$

$stateOccurrences : Q \to 2^{(\Delta \to 2^N)}$
for all $\delta = ((f, q_1 \dots q_n) \to q) \in \Delta$ **do**
  for $i \leftarrow 1$ **to** n **do**
    $stateOccurrences\,(q_i)\,(\delta) \to stateOccurrences\,(q_i)\,(\delta) \cup \{i\}$
  end for
end for
$eq : Q \to Q$, $eqClass : \left(Q \to \left((F, Q^*, Q) \to 2^N\right)\right) \to 2^Q$
$nonfinal \leftarrow q \in Q \setminus Q_f$, $final \leftarrow q \in Q_f$, $prevSize \leftarrow 0$
for all $q \in Q$ **do**
  if $q \in Q_f$ **then**
    $eq\,(q) \leftarrow final$
  else
    $eq\,(q) \leftarrow nonfinal$
  end if
end for
while true do
  for all $(q, D) \in stateOccurrences$ **do**
    $transitionPart : (F, Q^*, Q) \to 2^N$
    for all $(\delta = ((f, q_1 \dots q_n) \to q')\,, \{i_1 \dots i_k\}) \in D$ **do**
      for $j \leftarrow 1$ **to** k **do**
        $C \leftarrow \{q_1 \dots q_{j-1}, q_{j+1} \dots q_n\}$
        $transitionPart\,(f, C, eq\,(q')) \leftarrow transitionPart\,(f, C, eq\,(q')) \cup \{j\}$
      end for
    end for
    $eqClass\,(eq\,(q)\,, transitionPart) \leftarrow eqClass\,(eq\,(q)\,, transitionPart) \cup \{q\}$
  end for
  if $|eqClass| = prevSize$ **then**
    break
  end if
  $prevSize \leftarrow |eqClass|$, $eq \leftarrow \emptyset$
  for all $(K, \{q_1 \dots q_n\}) \in eqClass$ **do**
    for $i \leftarrow 1$ **to** n **do**
      $eq\,(q_i) \leftarrow q_1$
    end for
  end for
end while
$\Delta' \leftarrow \{(f, eq\,(q_1) \dots eq\,(q_n)) \to eq\,(q') : (f, q_1 \dots q_n) \to q' \in \Delta\}$
$Q' \leftarrow eq\,(Q)$, $Q'_f \leftarrow Q_f \cap Q'$
return A'

Kapitola 5

Převod stromového automatu na zásobníkový

Zásobníkový automat je na linearizovaných stromech silnější. Například pro stromový jazyk v prefixovém zápisu $a_2 b_1^n c_0 d_1^n e_0$ neexistuje stromový automat přijímající tento jazyk, narozdíl od zásobníkového automatu.

Protože je zásobníkový automat silnější, lze k danému stromovému automatu vytvořit zásobníkový automat přijímající stejný jazyk. Takový převod je závislý na zvolené linearizaci. Stromový jazyk s jeho linearizovaným protějškem musí být ekvivalentní, aby byli ekvivalentní i automaty přijímající je.

Jak bylo popsáno v sekci 1.6, prefixový a postfixový zápis ohodnoceného stromu jsou bijekce a jsou tedy vhodné pro převod automatů.

Způsob převodu stromového automatu na zásobníkový automat přijímající stromový jazyk v postfixovém zápisu je popsán v [3].

5.1 Návrh algoritmu

V [3] je popsán převod stromového automatu na bezkontextovou gramatiku a její následný převod na zásobníkový automat. Přímý převod je následující:

Definice 5.1. Buď $A = (Q, F, Q_f, \Delta)$ NFTA, pak *NPDA přijímající prázdným zásobníkem jazyk $L(A)$ v postfixovém zápisu s přidanou pravou zarážkou* $\dashv$ je $A' = (\{q\}, \Sigma, G, \delta, q, Z_0, \emptyset)$, kde $\Sigma = F \cup \{\dashv\}$, $G = Q \cup \{Z_0\}$ a δ je tvořena pravidly:

1. $\{\delta(q, a, q_1 \dots q_n) = (q, q') : \Delta(f, q_1 \dots q_n) = q'\}$
2. $\{\delta(q, \dashv, Z_0 q_f) = (q, \epsilon) : q_f \in Q_f\}$

Ekvivalentní definice pro automat přijímající finálním stavem je následující:

Definice 5.2. Buď $A = (Q, F, Q_f, \Delta)$ NFTA, pak *NPDA přijímající finálním stavem jazyk* $L(A)$ *v postfixovém zápisu s přidanou pravou zarážkou* $\dashv$ je $A' = (\{q, q_f\}, \Sigma, G, \delta, q, Z_0, \{q_f\})$, kde $\Sigma = F \cup \{\dashv\}$, $G = Q \cup \{Z_0\}$ a δ je tvořena pravidly:

1. $\{\delta(q, a, q_1 \dots q_n) = (q, q') : \Delta(f, q_1 \dots q_n) = q'\}$
2. $\left\{\delta\left(q, \dashv, Z_0 q'_f\right) = (q_f, \epsilon) : q'_f \in Q_f\right\}$

Rozdíl mezi oběma reprezentacemi je pouze v jejich množinách stavů a způsobem převodu finálních stavů původního automatu.

Výsledné automaty mají dva druhy pravidel. Pravidla z bodu 1. v definicích budou nazývána *neukončovací* a z bodu 2 *ukončovací*.

Z definic je také patrné, že každé pravidlo původního automatu má jedno ekvivalentní neukončovací pravidlo a každý koncový stav je ekvivalentní s jedním ukončovacím pravidlem.

Algoritmus tedy konstruuje zásobníkový automat následujícími kroky:

1. Množina stavů obsahuje nový libovolný stav q (resp. stavy q a q_f).
2. Počáteční stav je q.
3. Množina finálních stavů je prázdná (resp. obsahuje q_f).
4. Vstupní abeceda je abeceda původního automatu s přidaným symbolem pravé zarážky.
5. Symbol konce zásobníku je libovolný nový symbol Z_0.
6. Abeceda zásobníku obsahuje stavy původního automatu a symbol konce zásobníku Z_0.
7. Za každé pravidlo v původním automatu přidej neukončovací pravidlo podle definice 5.1.
8. Za každý koncový stav původního automatu přidej ukončovací pravidlo podle definice 5.1 (resp. 5.2).

5.2 Vlastnosti převedeného automatu

Stromový automat a převedený zásobníkový automat jsou ekvivalentní, proto pro každý algoritmus nad stromovým automatem lze zkonstruovat ekvivalentní algoritmus nad převedeným zásobníkovým automatem.

Převedený automat obsahuje pouze jeden nefinální stav, případně jeden finální. Ke zpracování řetězce se tedy používá pouze zásobník, který simuluje rekurzivní vyhodnocování stromového automatu.

Algoritmus 5 Převod NFTA na NPDA přijímající prázdným zásobníkem ekvivalentní jazyk v postfixovém zápisu

Vstup: NFTA $A = (Q, F, Q_f, \Delta)$
Výstup: NPDA $A' = (\{q\}, \Sigma, G, \delta, q, Z_0, \emptyset)$

```
Σ' ← F ∪ {⊣}
G ← Q ∪ {Z_0}
for all δ' = ((f, q_1 ... q_n) → q') ∈ Δ do
    δ ← δ ∪ {δ'}
end for
for all q_f ∈ Q_f do
    δ ← δ ∪ {((q, ⊣, Z_0 q_f) → (q, ϵ))}
end for
return A'
```

Algoritmus 6 Převod NFTA na NPDA přijímající finálním stavem ekvivalentní jazyk v postfixovém zápisu

Vstup: NFTA $A = (Q, F, Q_f, \Delta)$
Výstup: NPDA $A' = (\{q, q_f\}, \Sigma, G, \delta, q, Z_0, \{q_f\})$

```
Σ' ← F ∪ {⊣}
G ← Q ∪ {Z_0}
for all δ' = ((f, q_1 ... q_n) → q') ∈ Δ do
    δ ← δ ∪ {δ'}
end for
for all q'_f ∈ Q_f do
    δ ← δ ∪ {((q, ⊣, Z_0 q'_f) → (q_f, ϵ))}
end for
return A'
```

Počet symbolů odebraných ze zásobníku je pro každý symbol konstantní a je určen aritou ohodnoceného symbolu. Na zásobník se přidává vždy jeden symbol. Velikost přechodové funkce je proto omezena. Horní mez pro počet přechodových pravidel je:

$$|\delta| \leq |\Sigma| * |G|^{max(arity(\Sigma))} * |G| \tag{5.1}$$

Pokud je původní automat deterministický, je i převedený automat deterministický, protože jsou splněny všechny body z definice 1.42:

1. Levé a pravé strany původního pravidla a neukončovacího pravidla jsou ekvivalentní, proto pokud původní automat nemá více pravidel se stejnou levou stranou, nemá je i převedený automat. Ukončovací pravidla jsou pro každý finální symbol původního automatu unikátní, a protože se levé strany ukončovacích pravidel liší pouze tímto symbolem, neexistují dvě ukončovací pravidla se stejnou levou stranou.

2. Ukončující i neukončující pravidla mají fixní počet symbolů, proto pro daný symbol neexistují dvě pravidla, která se na levé straně liší délkou odebíraného zásobníkového řetězce.

3. Žádné pravidlo nečte ze vstupu prázdný řetězec.

5.3 Implementace

V automatové knihovně je implementován pouze zásobníkový automat přijímající koncovým stavem. Algoritmus je proto implementován podle pseudokódu 6. Soubor s implementací je `ToPostfixPushdownAutomaton.cpp`.

Jako nefinální stav je použita v knihovně standardizovaná instance stavu `label::InitialStateLabel::instance <DefaultStateType> ()`. Podobně pro finální stav je použita v knihovně standardizovaná instance finálního stavu `label::FinalStateLabel::instance<DefaultStateType>()`

Podobným způsobem existují v knihovně speciální symboly. Symbol zarážky je `alphabet::EndSymbol::instance<DefaultSymbolType>()` a symbol pro konec zásobníku je
`alphabet::BottomOfTheStackSymbol::instance<DefaultSymbolType>()`.

Kapitola 6

Minimalizace převedeného zásobníkového automatu

Převedený zásobníkový automat je ekvivalentní s jeho původním stromovým automatem, proto každý algoritmus pro stromový automat lze vytvořit pro převedený automat, včetně minimalizace.

Zásobníkový automat obsahuje kromě množiny stavů i množinu zásobníkových symbolů. Při pokusu o minimalizaci obecného zásobníkového automatu by se musely minimalizovat obě množiny. Díky vlastnostem převedeného automatu se ale není potřeba zabývat množinou stavů.

6.1 Návrh algoritmu

Algoritmus je popsán pro automat přijímající prázdným zásobníkem. Převedený automat přijímající koncovým stavem se z hlediska algoritmu liší pouze strukturou ukončovacích pravidel.

Algoritmus funguje stejným způsobem, jako pro stromový automat, s malými rozdíly. Místo se stavy se pracuje se zásobníkovými symboly. Protože má automat pouze jeden stav, nemusíme se zabývat minimalizací stavů.

V původním algoritmu jsou prvotní třídy ekvivalence finální a nefinální stavy. Finální stavy jsou v případě převedeného automatu ekvivalentní se symboly nacházejícími se v ukončujících pravidlech. Zvláštní třídou je také symbol konce zásobníku.

Ekvivalentní definice k 4.1 pro dělení tříd ekvivalence je:

Definice 6.1. Nechť $A = (\{q\}, \Sigma, G, \delta, q, Z_0, \emptyset)$ je převedený DPDA podle definice 5.1. Pro zásobníkové symboly α a α' a i-tou relaci ekvivalence P_i platí $\alpha P_{i+1} \alpha'$, pokud:

1. $\alpha P_i \alpha'$

2. $\delta_G(q, a, \alpha_1 \dots \alpha_{j-1}\alpha\alpha_{j+1} \dots \alpha_n)\, P_i \delta_G(q, a, \alpha_1 \dots \alpha_{j-1}\alpha'\alpha_{j+1} \dots \alpha_n)$, $\forall a \in \Sigma, \forall \alpha_1 \dots \alpha_{j-1}, \alpha_{j+1} \dots \alpha_n \in G$

Ekvivalentní struktura k 4.2 jednoznačně určující třídu ekvivalence pro symbol α je:

$$\left(eq_j(\alpha), \left\{\left(\left(f, \left(\alpha_1 \dots \alpha_{n-1}\right), eq_j\left(\alpha'\right)\right), \left\{i_1 \dots i_k\right\}\right)\right\}\right) \tag{6.1}$$

pro vstupní symbol f, zásobníkové symboly na levé straně pravidla $\alpha_1 \dots \alpha_{n-1}$, zásobníkový symbol na pravé straně pravidla α' a indexy výskytu $i_1 \dots i_k$.

V průběhu algoritmu se nemusí uvažovat ukončovací pravidla, protože nepřidávají na zásobník žádný symbol, a tím nikdy nezpůsobí dělení tříd ekvivalence.

Příklad 6.1. Uvažujme v dané iteraci algoritmu zásobníkové symboly α_0, α_1, α_2 a α_3, dvě třídy ekvivalence $E_1 = \{\alpha_0, \alpha_1\}$ a $E_2 = \{\alpha_3, \alpha_4\}$, a přechodová pravidla:

$$\begin{aligned}
(q, f, \alpha_0) &\to (q, \alpha_3) \\
(q, f, \alpha_1) &\to (q, \alpha_4) \\
(q, g, \alpha_0\alpha_0) &\to (q, \alpha_3) \\
(q, g, \alpha_0\alpha_1) &\to (q, \alpha_4) \\
(q, g, \alpha_1\alpha_0) &\to (q, \alpha_4) \\
(q, g, \alpha_1\alpha_1) &\to (q, \alpha_3)
\end{aligned}$$

Symboly α_0 a α_1 podle zadaných pravidel splňují podmínky z definice 6.1, a měly by proto patřit do stejné třídy ekvivalence i v další iteraci. Struktura podle 6.1 charakterizující třídu ekvivalence v další iteraci pro oba symboly vypadá následovně:

$$\left(E_1, \left\{\begin{array}{c} ((f, \{\}, E_2), \{1\}) \\ ((g, \{\alpha_0\}, E_2), \{1, 2\}) \\ ((g, \{\alpha_1\}, E_2), \{1, 2\}) \end{array}\right\}\right)$$

Pro symbol α_0 je první řádek struktury vytvořen z pravidla $(q, f, \alpha_0) \to (q, \alpha_3)$, druhý řádek z pravidla $(q, g, \alpha_0\alpha_0) \to (q, \alpha_3)$ a třetí řádek z pravidel $(q, g, \alpha_0\alpha_1) \to (q, \alpha_4)$ a $(q, g, \alpha_1\alpha_0) \to (q, \alpha_4)$.

Pro symbol α_1 je první řádek struktury vytvořen z pravidla $(q, f, \alpha_1) \to (q, \alpha_4)$, druhý řádek z pravidel $(q, g, \alpha_1\alpha_0) \to (q, \alpha_4)$ a $(q, g, \alpha_0\alpha_1) \to (q, \alpha_4)$, a třetí řádek z pravidla $(q, g, \alpha_1\alpha_1) \to (q, \alpha_3)$.

6.2 Implementace

Algoritmus je implementován podle pseudokódu 7. Implementace se nachází v souboru `Minimize.h`

Implementace je řešena stejným způsobem, jako u minimalizace stromového automatu. Místo stavů se pracuje se zásobníkovými symboly.

Pro mapování symbolů na symboly reprezentující třídu ekvivalence je použita `std::map <PushdownStoreSymbolType, PushdownStoreSymbolType>`

Struktura pro tvorbu tříd ekvivalencí vypadá následovně:

```
std::map<std::pair<PushdownStoreSymbolType,
                   std::map<
                         std::tuple<InputSymbolType,
                                    std::vector<
                                     PushdownStoreSymbolType>,
                                    PushdownStoreSymbolType>,
                         std::set<int>>>,
         std::set<PushdownStoreSymbolType> >
```

Algoritmus 7 Minimalizace převedeného redukovaného DPDA

Vstup: DPDA $A = (\{q\}, \Sigma, G, \delta, q, Z_0, \emptyset)$
Výstup: DPDA $A' = (\{q\}, \Sigma, G', \delta', q, Z_0, \emptyset)$

```
G_f ← ∅, symbolOccurrences : G → 2^(δ→2^N)
for all d = ((q, f, α_1 ... α_n) → (q, α)) ∈ δ do
  if d = (q, ⊣, Z_0 α') → (q, ϵ) then
    G_f ← G_f ∪ {α'}
  else
    for i ← 1 to n do
      stateOccurrences (α_i) (δ) → stateOccurrences (α_i) (δ) ∪ {i}
    end for
  end if
end for
eq : G → G, eqClass : (G → ((Σ, G*, G) → 2^N)) → 2^G
nonfinal ← α ∈ G \ G_f, final ← α ∈ G_f, prevSize ← 0
for all α ∈ G do
  if q ∈ Q_f then eq (α) ← final else eq (α) ← nonfinal end if
end for
while true do
  for all (α, D) ∈ symbolOccurrences do
    transitionPart : (Σ, G*, G) → 2^N
    for all (d = ((q, f, α_1 ... α_n) → (q, α')), {i_1 ... i_k}) ∈ D do
      for j ← 1 to k do
        tp = transitionPart (f, {α_1 ... α_{j-1}, α_{j+1} ... α_n}, eq (α'))
        tp ← tp ∪ {j}
      end for
    end for
    eqClass (eq (α), transitionPart) ← eqClass (eq (α), transitionPart) ∪
    {α}
  end for
  if |eqClass| = prevSize then
    break
  end if
  prevSize ← |eqClass|, eq ← {(Z_0, Z_0)}
  for all (K, {α_1 ... α_n}) ∈ eqClass do
    for i ← 1 to n do
      eq (α_i) ← α_1
    end for
  end for
end while
δ' ← {(q, f, eq (α_1) ... eq (α_n)) → (q, eq (α)) : (q, f, α_1 ... α_n) → (q, α) ∈ δ}
G' ← eq (G)
return A'
```

Kapitola 7

Testování

Testování probíhá na úrovni metod a také celých programů. Pro otestování funkčnosti metod jsou použity unit testy za pomocí frameworku CppUnit. Pro testování celých programů jsou použity testovací terminálové skripty.

7.1 Unit testy

Unit testy jsou použity pro každou implementovanou metodu. Vyhodnocují se po kompilaci dynamických knihoven. Kromě otestování algoritmu také indikují chyby při změnách v částech projektu, na kterých je daná metoda závislá.

V testu se vytváří objekty, které jsou použity při volání metody a následně se kontroluje výsledek, zda odpovídá očekávání.

Unit test redukce se nachází v souboru `trimTest.cpp`, testy minimalizace v souboru `minimizeTest.cpp` a testy převodů na zásobníkové automaty v souboru `FTAtoPDATest.cpp`.

7.2 Testovací skripty

Testovací skripty jsou psané pro Bash a používají programy knihovny. Vyhodnocují se po kompilaci celého projektu. Testují jak jednotlivé programy, tak celé výpočetní roury. Jako data využívají buď soubory ve složce `examples2/` nebo generované struktury z programu `arand2`.

Funkčnost programu `aminimize` testuje skript `tests.aminimize.sh`. Jako data dvojice souborů pro vstup a očekávaný výstup. Tyto soubory se nachází ve složce `examples2/automaton/` a jsou to všechny dvojice `<název>.xml` a `<název>.MIN.xml`. Porovnání provádí program `acompare2`.

Knihovna již obsahuje test pro algoritmy z arbologie. Jedním z takových algoritmů je vytvoření stromového automatu pro exaktní vyhledávání podstromů ve stromu. Výsledný automat se pak spouští na připravených příkladech i náhodných stromech. Redukcí a minimalizací vytvořeného automatu se

nezmění výsledek testu, proto jako další test je do roury přidána redukce a minimalizace. Test se nachází v souboru `tests.aarbology.sh`

```
void trimTest::testTrimDFTA() {
	automaton::DFTA < > automaton;

	std::vector<DefaultStateType> q;
	for (int i = 0; i <= 11; ++i) {
		DefaultStateType state (i);
		q.push_back(state);
		automaton.addState(state);
	}
	automaton.addFinalState(q[2]);
	automaton.addFinalState(q[11]);

	const std::ranked_symbol < > a ("a", 2);
	const std::ranked_symbol < > b ("b", 1);
	const std::ranked_symbol < > c ("c", 0);
	automaton.addInputSymbol(a);
	automaton.addInputSymbol(b);
	automaton.addInputSymbol(c);

	automaton.addTransition(c, {}, q[0]);
	automaton.addTransition(a, {q[0], q[0]}, q[1]);
	automaton.addTransition(b, {q[1]}, q[2]);
	automaton.addTransition(a, {q[3], q[4]}, q[5]);
	automaton.addTransition(b, {q[5]}, q[6]);
	automaton.addTransition(a, {q[2], q[2]}, q[7]);
	automaton.addTransition(a, {q[7], q[7]}, q[8]);
	automaton.addTransition(a, {q[9], q[9]}, q[10]);
	automaton.addTransition(a, {q[10], q[10]}, q[11]);

	automaton::DFTA<> trimed = automaton::simplify::Trim::trim(automaton);

	automaton::DFTA<> correct;
	correct.addState(q[0]);
	correct.addState(q[1]);
	correct.addState(q[2]);
	correct.addFinalState(q[2]);
	correct.addInputSymbol(a);
	correct.addInputSymbol(b);
	correct.addInputSymbol(c);
	correct.addTransition(c, {}, q[0]);
	correct.addTransition(a, {q[0], q[0]}, q[1]);
	correct.addTransition(b, {q[1]}, q[2]);

	CPPUNIT_ASSERT(trimed == correct);
}
```

Obrázek 7.1: Příklad CppUnit testu

Kapitola 8

Minimalizace dalších tříd zásobníkových automatů

Minimalizace obecného zásobníkového automatu není možná. Pro zásobníkové automaty je problém univerzality, tedy zda přijímají každý řetězec nad svou abecedou, nerozhodnutelný. Minimální automat pro takový jazyk lze triviálně sestrojit s jedním koncovým stavem a žádným zásobníkovým symbolem. Pokud by tedy existoval algoritmus minimalizace, byl by rozhodnutelný i problém univerzality.

K problému univerzality se také váže problém ekvivalence automatů. Ten je pro obecné automaty zásobníkové automaty také nerozhodnutelný, protože by se jinak dala univerzalita rozhodnout ekvivalencí vůči zmíněnému minimálnímu automatu. Jiná situace nastává pro deterministické automaty, pro které bylo v [4] dokázáno, že ekvivalence je rozhodnutelná.

Jedním z důvodů, proč zmíněné problémy jsou nerozhodnutelné, je fakt, že obecné zásobníkové automaty nemají omezenou přechodovou funkci, protože lze v pravidlu odebrat nebo přidat libovolně mnoho zásobníkových symbolů. Pro danou množinu stavů a zásobníkových symbolů proto může existovat nekonečně mnoho různých automatů.

Pokud máme třídu zásobníkových automatů, které mají omezenou přechodovou funkci a ekvivalence je pro ně rozhodnutelný problém, lze takové automaty minimalizovat. Zcela naivní algoritmus by v konečné množině automatů našel všechny ekvivalentní a vybral jeden s nejmenší velikostí reprezentace.

8.1 Existující výsledky

Podmínky omezenosti a rozhodnutelnosti ekvivalence splňuje třída visibly pushdown automatů, VPA zkráceně. VPA je zásobníkový automat, kde zásobníkové operace v každém pravidlu jsou omezené na nejvýše jeden symbol a vstupní abeceda je rozdělena na tři množiny symbolů podle povolené zá-

sobníkové operace, kdy lze buď pouze odebírat symboly ze zásobníku, pouze přidávat symboly na zásobník nebo se neprovádět žádnou zásobníkovou operaci.

VPA mají omezenou přechodovou funkci a lze je determinizovat a tedy i rozhodnout o ekvivalenci. Nad touto třídou již byla studována minimalizace v [5]. Protože je ukázáno, že minimalizace nedeterministických VPA patří do EXPTIME třídy složitosti a minimalizace deterministických VPA do NP, je největším problémem výpočetní náročnost. V článku jsou proto ukázány podtřídy VPA s polynomiálními algoritmy. V [6] se také zabývali aproximací minimalizace VPA.

8.2 Minimalizace jednostavového automatu

Automaty popsané v kapitole 5 také splňují omezenost a rozhodnutelnost ekvivalence. Minimalizace je navíc polynomiální vůči velikosti množiny zásobníkových symbolů, protože v nejhorším případě nastane $|G|$ dělení tříd ekvivalence a v každé iteraci se pro každý symbol podle rovnice 5.1 pracuje s polynomiálně mnoho přechodovými pravidly, protože počet zásobníkových symbolů přidávaných na zásobník je omezen aritou symbolů, která je konstatní.

Jedním z důvodů, proč na této třídě funguje minimalizace stejným způsobem jako u stromových automatů, je existence ukončovacích pravidel, která mají přesnou formu a podle kterých lze určit počáteční třídy ekvivalence. Mimo těchto pravidel a použití ϵ-přechodů nejsou na formu pravidel žádné restrikce. Cyklus dělení tříd ekvivalence proto může fungovat i na automatech bez přesně stanovených ukončovacích pravidlech. Zda lze zkonstruovat pro takové automaty počáteční třídy ekvivalence, a jakým způsobem, zůstává otevřenou otázkou.

Dosud se také uvažovaly pouze jednostavové automaty, kde se přidával na zásobník pouze jeden symbol. Další otevřenou otázkou proto je, zda lze podobný princip použít i pro automaty přidávající více symbolů na zásobník.

Závěr

Cílem této práce bylo navrhnout a implementovat algoritmy minimalizace pro stromové a převedené zásobníkové automaty.

Pro stromové automaty byl popsán a implementován algoritmus redukce, minimalizace a převodu na zásobníkové automaty. Algoritmus minimalizace byl poté upraven pro převedené zásobníkové automaty.

V práci se také diskutovalo o možném zobecnění navrženého algoritmu na větší třídu automatů. Z vlastností převedeného algoritmu vyplynulo, že lze minimalizovat stejným způsobem jako stromový automat, protože obsahuje jediný stav, přechodová pravidla přidávají na zásobník jeden symbol a také proto, že v automatu existují pravidla, nazývaná ukončovací, s pevně danou strukturou. Díky ukončovacím pravidlům lze vytvořit počáteční třídy ekvivalence zásobníkových symbolů, které algoritmus vyžaduje, a zbylé pravidla už nejsou strukturálně omezena. Otevřenou otázkou proto zůstává, zda lze vytvořit počáteční třídy ekvivalence u automatů bez přesně daných ukončovacích pravidel. Další otevřenou otázkou je, zda lze použít podobný princip u automatů přidávajících na zásobník více než jeden symbol.

Literatura

[1] Štěpán Plachý: *Automatová knihovna – Stromové automaty a algoritmy nad stromy*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2014.

[2] Comon, H.; Dauchet, M.; Gilleron, R.; aj.: Tree Automata Techniques and Applications. 2004.

[3] Janousek, J.; Melichar, B.: On regular tree languages and deterministic pushdown automata. *Acta Inf.*, ročník 46, 2009: s. 533–547.

[4] Sénizergues, G.: The Equivalence Problem for Deterministic Pushdown Automata is Decidable. In *ICALP*, 1997.

[5] Chervet, P.; Walukiewicz, I.: Minimizing Variants of Visibly Pushdown Automata. In *MFCS*, 2007.

[6] Heizmann, M.; Schilling, C.; Tischner, D.: Minimization of Visibly Pushdown Automata Using Partial Max-SAT. In *TACAS*, 2017.

Příloha A

Seznam použitých zkratek

NFTA Nedeterministický konečný stromový automat (Nondeterministic Finite Tree Automaton)

DFTA Deterministický konečný stromový automat (Deterministic Finite Tree Automaton)

NPDA Nedeterministický zásobníkový automat (Nondeterministic PushDown Automaton)

DPDA Deterministický zásobníkový automat (Deterministic PushDown Automaton)

VPA Visibly Pushdown Automat

BU Zdola nahoru (Bottom-Up, typ stromového automatu)

TD Shora dolů (Top-Down, typ stromového automatu)

XML eXtensive Markup Language

Příloha B

Uživatelská příručka

B.1 Požadavky

Pro úspěšnou kompilaci je vyžadováno:

- program `make` ve verzi alespoň 3.9
- program `g++` ve verzi alespoň 4.8 nebo `clang++` ve verzi alespoň 3.5
- knihovna `libcppunit-dev`
- knihovna `libtclap-dev`
- knihovna `libxml2-dev`

B.2 Instalace

Zdrojové kódy se zkompilují příkazem

```
$ make release
```

Jednotlivé aplikace se pak nacházejí v adresáři `bin-release/`. Pro spouštění programů je potřeba mít tuto složku v proměnné `PATH`, případně aplikace spouštět z této složky.

Uvedených příkladech jsou cesty souborů uváděné relativně vůči adresáři `bin-release/`.

B.3 Redukce deterministického bottom-up konečného stromového automatu programem atrim2

Program očekává XML reprezentaci deterministického stromového automatu buď na vstupu nebo, s přepínačem, v souboru.

Přepínače ovlivňující redukci automatu jsou:

- `-u, --useless` – zvolí odstranění zbytečných stavů
- `-r, --unreachable` – zvolí odstranění nedosažitelných stavů
- `-i <file>, --input <file>` – specifikuje soubor s automatem

Příklad B.1. `$ ./atrim2 -r -u -i ../examples2/automaton/DFTA1.xml`

B.4 Minimalizace bottom-up konečného stromového automatu a převedeného zásobníkového automatu programem aminimize2

Program očekává XML reprezentaci redukovaného deterministického automatu buď na vstupu případně, s přepínačem, v souboru.

Přepínač je:

- `-i <file>, --input <file>` – specifikuje soubor s nedeterministickým automatem

Příklad B.2. `$ ./aminimize2 -i ../examples2/automaton/PostfixDPDA1.xml`

Příklad B.3. `$ ./aminimize2 -i ../examples2/automaton/PostfixDPDA1.xml`

B.5 Převod bottom-up konečného stromového automatu na zásobníkový automat programem aconversions2

Program očekává XML reprezentaci formalismu buď na vstupu nebo, s přepínačem, v souboru.

Přepínače ovlivňující převod jsou:

- `-t <pda>, --target <pda>` – zvolí cílový formalismus, v tomto případě `pda`
- `-i <file>, --input <file>` – specifikuje soubor s automatem

Příklad B.4. `$ ./aconversions2 -t pda -i ../examples2/automaton/DFTA1.xml`

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `automata-library` ..... zdrojové soubory Automatové knihovny
- `text`
  - `DP_Plachy_Stepan_2017.pdf` ..... text práce ve formátu PDF
  - `src` ..... zdrojové soubory práce ve formátu LaTeX