VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ PRO PRÁCI S BÜCHI AUTOMATY

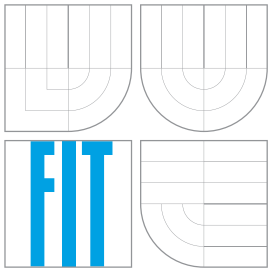
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR SCHINDLER

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ PRO PRÁCI S BÜCHI AUTOMATY

TOOL FOR BÜCHI AUTOMATA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. PETR SCHINDLER
AUTHOR

VEDOUCÍ PRÁCE — Mgr. ADAM ROGALEWICZ, Ph.D.
SUPERVISOR

BRNO 2013

Abstrakt

Tato práce zpracovává problematiku Büchiho automatů a představuje knihovnu, umožňující provádět základní operace nad těmito automaty. V práci jsou uvedeny základy teorie automatů. Ty jsou použity při popisu konečných automatů, mezi které patří Büchiho automaty, jejichž popis tvoří hlavní část zde uvedené teorie. Znalost jejich vlastností je důležitá pro pochopení algoritmů, které s nimi pracují. Dále jsou tyto algoritmy představeny i s detailním vysvětlením. Následuje návrh datových struktur formátu, v jakém jsou automaty ukládány na disk. Stěžejní část práce se věnuje implementaci knihovny a pomocných skriptů. Detailně jsou zde popsány důležité nebo zajímavé části implementace jednotlivých metod. Závěr práce je věnován testování funkčnosti knihovny.

Abstract

This thesis elaborates the Büchi automata theory and intorduces a library that enables working with them. Basics of the automata therory is explained. The main part of this work is focused on Büchi automata, which belong to finite automata. The main properties of Büchi automata are explained and proved. The knowledge of those properties is important for understanding the algorithms mentioned in this work. The next part describes those algorithms and explains their principles in details. The design of library is described in the next part of this work. Main part is aimed at the implementation of the library and auxiliary scripts. The most interesting and important parts of methods are described in detail. Closing part describes testing of library functionality.

Klíčová slova

Büchiho automat, konečný automat, knihovna pro Python, operace nad Büchiho automaty.

Keywords

Büchi automata, final state automata, Python library, operations over Büchi automata.

Citace

Petr Schindler: Nástroj pro práci s Büchi automaty, diplomová práce, Brno, FIT VUT v Brně, 2013

Nástroj pro práci s Büchi automaty

Prohlášení

Prohlašuji, že jsem tento diplomový projekt vypracoval samostatně pod vedením pana Mgr. Adama Rogalewicze, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Schindler
22. května 2013

Poděkování

Rád bych poděkoval Mgr. Adamu Rogalewiczovi, Ph.D., za jeho odborné rady a pomoc při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Teoretická informatika je velice zajímavým odvětvím informačních věd. Její výsledky nejsou vidět okamžitě, avšak dopad na obor je obrovský. Bez teoretických základů by bylo jen těžce představitelné, že by počítače vůbec existovaly. Chceme-li vytvářet efektivní algoritmy na efektivních strojích, potřebujeme vždy něco z teoretické informatiky, do níž patří i teorie složitosti a logika. Velmi významným odvětvím teoretické informatiky je teorie automatů a gramatik. Automaty lze modelovat různé systémy, mimo jiné i samotné operační systémy či jejich části. Popis systému pomocí automatu nám umožní jej studovat, hledat v něm chyby nebo zkoumat, zda odpovídají požadované specifikaci.

Tato práce se zabývá Büchiho automaty, jež patří do kategorie konečných automatů. Jejich důležitá vlastnost je, že pracují s nekonečnými slovy. To z nich dělá velice dobré prostředky pro modelování systémů, jejichž běh (v ideálním případě) nikdy nekončí. Tohoto se využívá v procesu tak zvaného *model checkingu* (český výraz kontrola modelu se používá velmi málo) - jedné z technik pro formální verifikaci systémů. Způsobů, jak zjišťovat, jestli zadaný systém odpovídá dané specifikaci, je spousta. Jedna z možností je použití Büchiho automatu a to následovně. Vytvoříme Büchiho automat B modelující systém a druhý B_{spec} pro specifikaci problému. Pokud verifikovaný systém odpovídá specifikaci, pak platí, že jazyk tvořený B je podmnožinou jazyka automatu B_{spec}.

Tato práce má za cíl vytvoření knihovny a ukázkového programu pro práci s Büchiho automaty. V předchozím odstavci je vidět, že taková práce může nalézt uplatnění. Tuto práci jsem si zvolil, protože mám rád teoretické problémy a jejich převod do praxe.

Práce je členěna do několika částí. V kapitole 2 jsou uvedeny poznatky z oblasti konečných automatů a to od obecného popisu oboru teorie automatů přes konečné automaty po Büchiho automaty. Jejich důležité vlastnosti jsou zde představeny i s důkazy, které většinou tvoří základ pro tvorbu algoritmů nad těmito automaty. Kapitola 3 představuje jednotlivé algoritmy. Pro každou metodu je mimo algoritmu samotného uveden též princip, na kterém je metoda založena. Detailněji je popsán výpočet doplňku, jehož konstrukce obsahuje pojmy, které je nutné vysvětlit. Návrh knihovny je uveden v kapitole 4. Začátek kapitoly se věnuje očekávanému použití knihovny. Dále je zde popsán návrh datových struktur, které jsou při implementaci použity pro modelování Büchiho automatů. Na závěr je uveden způsob zápisu automatů do souboru.

Popis implementace knihovny je uveden v kapitole 5. Většina kapitoly je věnována třídě `Buchi`, jejím atributům a hlavně jejím metodám. Implementace jednotlivých metod je probrána detailně se zaměřením na ty části, jež jsou implementačně zajímavé či jinak důležité. Dále jsou zde uvedeny popisy třídy modelující Kripkeho strukturu a skriptů, jež využívají vzniklou knihovnu, případně slouží k jejímu testování. Závěr této práce se věnuje

testování (kapitola 6). Je zde popsáno, jakým způsobem byla vytvořena testovací sada automatů a jak probíhalo testování samotné.

Kapitola 2

Konečné automaty

2.1 Teorie formálních jazyků a automatů

Obecně je oblast teoretické informatiky velmi obtížné vymezit. Teoretická (matematická) informatika zkoumá matematické zákonitosti, které mají využití při zpracování informací. Jako hlavní disciplíny teoretické informatiky uvádí [7] teorii formálních jazyků a automatů, teorii vyčíslitelnosti a složitosti a logiku.

Jedna z klasických a velmi důležitých disciplín teoretické informatiky je teorie automatů. **Teorie automatů** je věda zabývající se popisem matematických objektů zvaných automaty a řešením problémů s jejich využitím. Slovo automat pochází z řečtiny a ve volném překladu znamená „samohybný“ (z řeckého *automatos*). Teorie automatů vznikla ve třicátých letech 20. století pro účely logiky a snažila se najít konečný popis nekonečných objektů. Mezi zakladatele oboru a první badatele patří Turing, Kleene, Post, Church či Markov. Turing roku 1936 vytvořil definici stroje, kterému se dnes říká Turingův stroj a který je důležitou součástí teoretické informatiky. Další rozvoj je spojován s mikrobiology McGullcikem a Pittsem (1943). Na konci padesátých let 20. století byly položeny základy formální teorie konečných automatů, za níž stáli Huffmann a Kleene. [4]

Automaty patří k výpočetním modelům. Tyto modely se snaží zachycovat a simulovat různé problémy vypočitatelnosti a rozhodnutelnosti. Výpočetní modely lze rozdělit na tři základní typy podle vyjadřovací síly na [10]:

(i) modely s konečným množstvím paměti,

(ii) modely s konečným množstvím paměti se zásobníkem,

(iii) modely bez omezení.

Bezesporu nejdůležitějšími automaty spadajícími do první kategorie jsou konečné automaty, mezi které patří např. Büchiho automaty. Těmito automaty se zabývá tato práce a jsou popsány blíže v kapitole 2.1.3. Dalším příkladem patřícím do první kategorie jsou regulární výrazy. Druhá kategorie obsahuje automaty se zásobníkem. Do třetí skupiny patří již zmiňovaný Turingův stroj, μ-rekurzivní funkce nebo též λ-kalkul. [10]

Nedílnou součástí teorie automatů je **teorie formálních jazyků** spadající ke složitějším teoretickým disciplínám informatiky. Souvislost s výše uvedeným rozdělením a počátky novodobé historie teorie formálních jazyků jsou spojovány s americkým matematikem Noamem Chomskym, který vytvořil matematický model gramatiky jazyka. Podle [7] tato nově vytvořená teorie jazyka pracuje se dvěma duálními matematickými entitami a to

gramatikou a automatem. Automat zde představuje abstraktní matematický stroj, který dovede identifikovat strukturu vět formálního jazyka. Gramatika umožňuje strukturu vět formálního jazyka popsat. Tak zvaná Chomského hierarchie formálních jazyků dělí třídy jazyků popsaných gramatikami podle jejich rostoucí složitosti na:

(i) pravé lineární gramatiky,

(ii) bezkontextové gramatiky,

(iii) neomezené gramatiky.

Tyto jednotlivé typy jazyků přímo korespondují k rozdělení výpočetních modelů z předchozího odstavce. Tzn. např. pravou lineární gramatiku, respektive jazyk jí popisovaný, lze vyjádřit regulárními výrazy či konečnými automaty [10]. Jiná literatura [20] doplňuje čtvrtý typ, tzv. kontextové gramatiky, a zařazuje jej mezi bezkontextové a neomezené gramatiky. Jazyk, který kontextové gramatiky popisují, lze popsat tzv. lineárně omezeným automatem).

2.1.1 Základní pojmy

Základními pojmy používanými v teorii automatů (a teorii formálních jazyků) jsou abeceda a řetězec. **Abeceda** je neprázdná množina určitých prvků, které nazýváme **symboly** či znaky jazyka. Abeceda může potencionálně obsahovat nekonečné množství znaků. Pro potřeby této práce si vystačíme pouze s konečnými množinami. Příkladem abecedy může být latinka, která obsahuje malé a velké znaky *a* až *z*. Dalším příkladem je abeceda binární obsahující pouze dva prvky *1* a *0*.

Řetězec (též slovo či věta) nad danou abecedou je posloupnost (potencionálně nekonečná) symbolů abecedy. Speciálním případem je prázdný řetězec, který se většinou označuje písmenem ϵ. Formálně můžeme definovat řetězec nad abecedou Σ např. iterativně a to následovně [20]:

1. Prázdný řetězec ϵ je řetězec nad abecedou Σ.
2. Je-li x řetězec nad Σ a $a \in \Sigma$, pak i xa je řetězec nad abecedou Σ.
3. y je řetězec nad Σ pouze tehdy, je-li vytvořen aplikováním pravidel 1 a 2 (počet opakování může být potencionálně nekonečný).

Množina všech konečných řetězců nad abecedou Σ se označuje jako Σ^*. Pro tuto práci jsou důležitější nekonečná slova, která lze definovat jako zobrazení $r : \mathbb{N} \to \Sigma$. Nekonečná slova se též často označují jako ω-slova a množina všech takových slov je označována jako Σ^ω.

Množina řetězců nad danou abecedou se nazývá **jazykem**. Formálně je tedy definován jazyk L jako $L \subseteq \Sigma^*$.

K teorii automatů se pojí pojem **stav**. Stavem nějakého systému nazýváme jeho popis v určitém okamžiku. Stav by měl poskytovat veškeré relevantní informace. **Přechodem** nazýváme změnu jednoho stavu na druhý a může nastat sám od sebe či jako odpověď na vnější podmět (vstup).

2.1.2 Konečné automaty nad konečnými slovy

Významnou disciplínou teoretické informatiky je teorie konečných automatů. Základním pojmem této problematiky je konečný automat. Obecně lze **konečný automat** definovat jako abstraktní model konečně stavových systémů [19], neboli virtuální systém s konečným počtem stavů [9].

Konečné automaty lze rozdělit podle různých hledisek [9]. Existují dvě základní rozdělení. Konečné automaty lze rozdělit na:

- konečné automaty bez výstupu neboli dle [1] rozpoznávací automaty (akceptory),
- konečné automaty s výstupem.

Podle jiného kritéria lze konečné automaty dělit na:

- konečné automaty deterministické,
- konečné automaty nedeterministické.

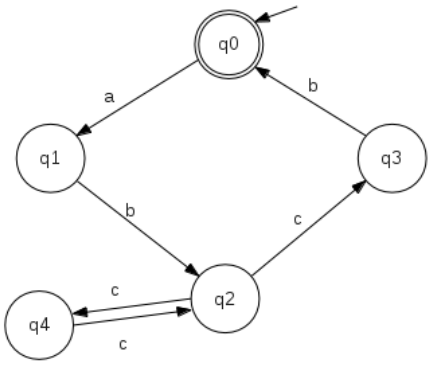
Definice 2.1. *Formální definice konečného automatu A je pětice $A = (Q, \Sigma, \delta, s_0, F)$, kde:*

- *S je konečná množina stavů,*
- *Σ je abeceda,*
- *δ je přechodová funkce,*
- *$s_0 \in Q$ je počáteční stav,*
- *$F \subseteq Q$ je množina konečných (někdy též akceptujících) stavů.*

Podle podoby přechodové funkce určujeme, zda je automat deterministický či nedeterministický. U deterministického automatu můžeme v každé chvíli provádění automatu určit, který stav bude následovat při daném vstupu. Přechodová funkce je tedy zobrazení $\delta : Q \times \Sigma \to Q$. Na druhou stranu, pokud máme nedeterministický automat, pak můžou existovat stavy, ze kterých při stejném vstupu lze přejít do několika odlišných stavů. Nedeterministická přechodová funkce vypadá následovně $\delta : Q \times \Sigma \times Q$. Přechodovou funkci je též možné zapsat ve formě funkce $\delta : Q \times \Sigma \to 2^Q$. U konečných automatů s konečným vstupem je vyjadřovací síla obou automatů stejná (důkaz v [15]), tudíž mezi nimi lze převádět.

Konečný automat lze zobrazit též jako orientovaný graf $G = (V, E)$ s ohodnocením hran $v : E \to \Sigma$. Vrcholy V korespondují se stavy automatu. Množina hran $E = \{(q, q') \mid \exists a \in \Sigma : (q, a, q') \in \delta\}$. Ohodnocovací funkce v je pak dána předpisem $\forall q, q' \in Q, \forall a \in \Sigma : (q, a, q') \in \delta \Rightarrow v((q, q')) = a$. Ukázka grafové reprezentace konečného automatu je na obrázku 2.1. Vstupní stav je označen šipkou, konečný stav je vyznačen dvojitým kruhem.

Princip fungování konečného automatu lze popsat následovně. Práce automatu začíná v počátečním stavu. Následovně na



Obrázek 2.1: Příklad grafové reprezentace konečného automatu.

vstup přicházejí symboly vstupní abecedy a
konečný automat na jejich příchod reaguje
přechodem do následujícího stavu. Posloupnost stavů je dána přechodovou funkcí. Automat pracuje tak, že pokud je ve stavu $q \in Q$, na vstupu má symbol $a \in \Sigma$ a přechodová funkce δ obsahuje trojici (q, a, q'), změní se stav automatu na q'. Pokud je stroj deterministický, existuje nejvýše jedno takové q'. Po načtení celého vstupu práce automatu končí.

Tento proces nyní definuji formálně. Dvojici $C = (q, w)$ z $Q \times \Sigma^*$ nazýváme **konfigurací automatu**. Počáteční konfigurace je $C_0 = (s_0, w)$, kde w je přijímané slovo.

Jednoduchý přechod automatu A se značí $\vdash_A$ a je binární relací na množině všech konfigurací. Jestliže $(q, a, q') \in \delta$, pak $(q, aw) \vdash_A (q', w)$ pro všechna $w \in \Sigma$. $\vdash_A^*$ pak značíme tranzitivní a reflexivní uzávěr relace $\vdash_A$. Pro dvě konfigurace C_0 a C_n zřejmě platí, že $C_0 \vdash_A C_n$ právě tehdy, když existuje posloupnost přechodů $C_0 \vdash_A C_1 \vdash_A C_2 \vdash_A \ldots \vdash_A C_n$.

Výpočet automatem A je konečná sekvence konfigurací $C_0, C_1, \ldots, C_n$ pro nějaké $n \geq 0$ taková, že:

- C_0 je počáteční stav,
- C_n má tvar (q, ϵ), kde $q \in Q$ (ϵ znamená, že celý vstup byl přečten),
- $C_0 \vdash_A^* C_n$.

Mějme slovo $w \in \Sigma^*$. Automat A přijímá slovo w, právě tehdy když $(s_0, w) \vdash_A^* (q, \epsilon)$, kde $q \in F$. Konfigurace (q, ϵ) pro $q \in F$ se nazývá akceptující stav. Pokud se nelze dostat z počáteční konfigurace do akceptující, je slovo zamítnuto.

Jazyk přijímaný automatem A je pak množina všech řetězců, které daný automat přijímá, formálně $L(A) = \{w \mid (s_0, w) \vdash_A^* (q, \epsilon), q \in F\}$. $L(A)$ se též nazývá regulární množina a jazyk, který tvoří se jmenuje regulární jazyk. S těmito množinami budeme pracovat v následující kapitole.

2.1.3 Konečné automaty nad nekonečnými slovy

V předchozí kapitole jsme se zabývali konečnými automaty nad konečnými slovy. V některých případech je ovšem užitečné umět pracovat s nekonečnými řetězci. Můžeme například chtít modelovat operační systém nebo chování dvou konkurenčních systémů, případně nějaký hardware. Všechny tyto systémy by měly v ideálním případě běžet bez chyby do nekonečna. Pokud bychom chtěli modelovat takový systém, je nezbytné definici konečných automatů rozšířit tak, aby mohly přijímat nekonečné řetězce (které simulují nekonečný sled událostí systému).

Automaty pracující s nekonečnými slovy přijímají jazyky $L \subseteq \Sigma^\omega$. Aby konečné automaty mohly akceptovat tyto jazyky, musí být upravena podmínka pro přijetí slov. Julius Richard Büchi v [3] definoval akceptační podmínku konečných automatů tak, že přijímají ω-jazyky. Automatům s touto podmínkou se říká **Büchiho automaty**. V další části jsou krátce zmíněny i jiné akceptační podmínky.

Nedeterministický Büchiho automat B je pětice $B = (Q, \Sigma, \delta, S, F)$, kde jednotlivé složky mají podobný význam, jako u konečných automatů nad konečnými slovy (vizte stranu 6), s tím rozdílem, že $S \subseteq Q$ je množina počátečních stavů a stavy z F nenazýváme konečné, ale akceptující. Pro deterministický Büchiho automat pak platí, že S má pouze jeden prvek s_0 a přechodová funkce je ve tvaru $\delta : (Q \times \Sigma) \to Q$.

Pojmy konfigurace a přechod můžeme též použít z předchozí kapitoly (vizte stranu 7), s tím rozdílem, že se zde pracuje s nekonečnými slovy $w \in \Sigma^\omega$. Pro Büchiho automaty se více než přechod používá pojem **běh**.

Definice 2.2. *Mějme slovo $w \in \Sigma^\omega$. Běh automatu B nad slovem w je totální funkce $\rho : \mathbb{N} \to Q$, taková, že*

- *první stav je počáteční, to znamená, že $\rho(0) \in S$*
- *postupuje dle přechodové funkce, to jest $\forall i \in \mathbb{N}_0 : (\rho(i), w(i), \rho(i+1)) \in \delta$. $w(i)$ symbolizuje i-tý znak slova w.*

Běh ρ automatu B je totožný s nějakou cestou jeho grafu z počátečního stavu, kde hrany jsou postupně ohodnoceny znaky přijímaného slova w. Dále definujeme $inf(\rho)$ jako množinu stavů, které se vyskytují v ρ nekonečně často. [5]

Běh ρ automatu B je akceptující, pokud platí, že $inf(\rho) \cap F \neq \emptyset$, tedy pokud je v množině nekonečně se opakujících stavů běhu ρ alespoň jeden akceptující stav z F.

Pro některé aplikace se ovšem hodí mít akceptační podmínku definovanou odlišně. Nyní představím tři varianty Büchiho automatů. Tyto automaty mají stejnou vyjadřovací sílu jako nedeterministické Büchiho automaty. Každá z těchto variant má odlišně definovanou množinu akceptujících stavů a podmínku pro přijetí slova. Nyní tyto tři varianty krátce představím.

- Mullerův automat: Akceptační množina $F \subseteq 2^Q$. Běh ρ je přijímající právě tehdy, pokud $inf(\rho) \in F$.
- Streettův automat: $F \subseteq 2^Q \times 2^Q$. Běh ρ je přijat, pokud $\forall (F_1, F_2) \in F$ platí, že $inf(\rho) \cap F_1 \neq \emptyset \vee inf(\rho) \cap F_2 = \emptyset$.
- Rabinův automat: $F \subseteq 2^Q \times 2^Q$. Běh ρ je přijat, pokud $\exists (F_1, F_2) \in F$ pro kterou platí, že $inf(\rho) \cap F_1 = \emptyset \vee inf(\rho) \cap F_2 \neq \emptyset$.

Tyto varianty jsou zajímavé například tím, že pro ně existují deterministické varianty se stejnou vyjadřovací silou. Což, jak uvidíme později, pro Büchiho automaty neplatí.

Významným rozšířením Büchiho automatu je tzv. **zobecněný Büchiho automat**. Tato varianta má následující akceptační podmínku. Množina $F \subseteq 2^Q$. Běh ρ zobecněného Büchiho automatu je přijímající, pokud pro všechny $P_i \in F$ platí, že $inf(\rho) \cap P_i \neq \emptyset$. Tato varianta se využívá například pro převod z LTL [5].

Definice 2.3. *Jako ω-jazyk se označuje jazyk přijímaný nedeterministickým Büchiho automatem, tedy množina $L(B) = \{w \in \Sigma^\omega \mid$ existuje akceptující běh B nad $w\}$*

Tyto jazyky mají mnohé společného s regulárními jazyky (které přijímají konečné automaty nad konečnými slovy). Mějme slovo $w = a_0 a_1 \dots a_{n-1}$ nad abecedou Σ. Značme $q \underset{w}{\to} q'$, pokud existuje posloupnost $q_0, q_1, \dots, q_n$ taková, že $q_0 = q$ a $(q_i, a_i, q_{i+1}) \in \delta \ \forall i < n$ a $q_n = q'$. Budiž $W_{ss'}$ množina všech takových slov, které načteme při přechodu ze stavu q do stavu q' automatu B, tedy formálně $W_{ss'} = w \in \Sigma \mid q \underset{w}{\to} q'$. Pak každý z konečně mnoha jazyků W'_{qq} je regulární. Z toho, jak je definována podmínka pro přijetí slova Büchiho automatem, je tedy každý ω-jazyk rozpoznávaný automatem B [18]:

$$L(B) = \cup_{s_0 \in S} \cup_{q \in F} W_{q_0 s} (W_{qq})^\omega$$

.

Tvrzení 2.1. *Büchiho automat rozpoznává ω-jazyk $L \subseteq \Sigma^\omega$ právě tehdy, když L je konečné sjednocení množin UV^ω, kde $U, V \in \Sigma^*$ jsou regulární množiny konečných slov (navíc lze předpokládat, že $VV \subseteq V$).*

Důkaz zleva doprava je zřejmý z předchozího odstavce. Pro důkaz opačného směru ukážeme, že platí následující tvrzení:

Lemma 2.2. *(i) Pokud $V \subseteq \Sigma^*$ je regulární množina, pak B rozpoznává V^ω.*

(ii) Pokud $U \subseteq \Sigma^*$ je regulární množina a $L \subseteq \Sigma^\omega$ je jazyk rozpoznávaný Büchiho automatem B, pak i UL je jazyk přijímaný B.

(iii) Pokud $L_1, L_2 \subseteq \Sigma^\omega$ jsou jazyky rozpoznatelné Büchiho automaty, pak jsou i jazyky $L_1 \cup L_2$ a $L_1 \cap L_2$ jazyky přijímané Büchiho automaty.

Důkaz. (i) Protože $V^\omega = (V - \{\epsilon\})^\omega$, můžeme předpokládat, že V neobsahuje prázdný řetězec. Mějme automat A, který přijímá V. Předpokládejme, že do jeho počátečního stavu nevede žádný přechod. Z automatu A můžeme vytvořit Büchiho automat B, který přijímá V^ω tak, že přidáme přechod (q, a, s_0) za každý přechod (q, a, q'), kde $q' \in F$ a označíme s_0 za jediný akceptující stav automatu B.

(ii) Dokážeme obdobně. Předpokládejme konečný automat $A = (Q_A, \Sigma, \delta_A, s_0, F_A)$ přijímající U a Büchiho automat $B = (Q_B, \Sigma, \delta_B, S, F_B)$ přijímající L a předpokládejme, že $Q_A \cap Q_B = \emptyset$. Automat akceptující UL pak je $B' = (Q_A \cup Q_B, \Sigma, \delta', S', F_B)$, kde

- $\delta' = \delta_A \cup \delta_B \cup \{(q, a, q') \mid q' \in S \wedge \exists q_f \in F_A (q, a, q_f) \in \delta_A\}$,
- $S' = s_0$ pokud platí, že $s_0 \notin F_A$, jinak $S' = s_0 \cup S$.

(iii) Důkaz opět provedeme sestavením automatů B' pro sjednocení a B'' pro průnik. Mějme automaty $B_1 = (Q_1, \Sigma, \delta_1, S_1, F_1)$ a $B_2 = (Q_2, \Sigma, \delta_2, S_2, F_2)$ a předpokládejme, že $Q_A \cap Q_B = \emptyset$. Protože se jedná o nedeterministické automaty, které mohou mít více počátečních stavů, je sestrojení automatu B' jednoduché, stačí sjednotit jednotlivé složky a tak dostaneme $B' = (Q_1 \cup Q_2, \Sigma, \delta_1 \cup \delta_2, S_1 \cup S_2, F_1 \cup F_2)$. Sestrojení automatu pro průnik dvou ω-jazyků je mírně složitější. Vytvoříme automat, který bude sledovat dosažení akceptujícího stavu prvního automatu, následovně dosažení akceptujícího stavu druhého automatu a atd. Automat přijímající průnik tedy bude ve tvaru $B'' = ((Q_1 \times Q_2 \times \{0, 1, 2\}, \Sigma, \delta'', S_1 \times S_2 \times \{0\}, Q_1 \times Q_2 \times \{2\}))$. Do přechodové funkce patří prvky $((q_1, q_2, x), a, (q_1', q_2', y))$ právě tehdy, když platí následující pravidla:

- Nový přechod kopíruje přechody obou automatů, tedy $(q_1, a, q_1') \in \delta_1$ a $(q_2, a, q_2') \in \delta_2$.
- Třetí komponenta stavu závisí na navštívení akceptujících stavů B_1 a B_2. Tedy platí:
  - Pokud $x = 0$ a $q_1' \in F_1$, pak $y = 1$.
  - Pokud $x = 1$ a $q_2' \in F_2$, pak $y = 2$.
  - Pokud $x = 2$, pak $y = 0$.
  - Jinak $x = y$.

Třetí komponenta stavu reflektuje, které konečné stavy jsme již potkali. Pokaždé, když se dostaneme do stavu $(q, r, 2)$ víme, že jsme navštívili jak akceptující stav z B_1, tak i akceptující stav z B_2. Protože akceptujících stavů obou automatů je konečně mnoho, můžeme si být jistí, že pokud nekonečně krát navštívíme některý z akceptujících stavů B'', projdeme i nekonečně častokrát některým z akceptujících stavů B_1 i B_2. □

Büchiho automaty, na rozdíl od automatů pracujících s konečnými slovy, mají jednu nepříjemnou vlastnost. Vyjadřovací síla deterministických Büchiho automatů je ostře menší, než síla automatů nedeterministických. Následuje důkaz převzatý z [15]:

Tvrzení 2.3. *Existuje takový jazyk, pro který lze sestrojit nedeterministický Büchiho automat, avšak neexistuje žádný deterministický Büchiho automat, který by jej přijímal.*

Důkaz. Dokážeme, že žádný deterministický Büchiho automat nepřijímá jazyk $L = \{w \in \{a,b\}^\omega \mid$počet výskytů písmene b v w je konečný$\}$. Předpokládejme, že takový automat B existuje. Každý řetězec jazyka L má podobu wa^ω, kde w je konečný řetězec z $\{a,b\}^*$. Což je zřejmě pravda, protože všechny takové řetězce mají konečný počet písmene b. Odstraníme z B všechny stavy, které jsou nedosažitelné z počátečního stavu. Nyní vybereme libovolný stav q z B. Protože q je dosažitelný z počátečního stavu, musí existovat alespoň jedno konečné slovo, které postupným prováděním B dosáhne stavu q. Tento řetězec nazveme w. Z předchozího vyplývá, že slovo wa^ω patří do jazyka L a tedy musí být akceptováno automatem B. Aby ho B přijalo, musí existovat alespoň jeden akceptující stav q_a, který se vyskytuje nekonečně často při běhu B nad slovem wa^ω. Tento akceptující stav musí být tedy dosažitelný z q (což je stav B v momentu, kdy jsme načetli slovo w) konečným počtem načtení písmene a (toto je zajisté pravda, neboť máme pouze konečný počet stavů), tento počet nazveme a_q. Vypočítáme a_q pro všechny stavy q z B. Maximální hodnoty všech a_q pak nazveme m.

Nyní můžeme ukázat, že takový automat B přijímá též řetězec $(ba^m)^\omega \notin L$. Protože B je deterministický automat, je přechodová funkce definována pro všechny dvojice (stav, vstup), tudíž stroj B musí běžet do nekonečna na všech řetězcích, včetně $(ba^m)^\omega$. Z předchozího odstavce víme, že z každého stavu existuje slovo obsahující nejvíce m znaků a, které může automat B posunout do akceptujícího stavu. Tedy po každém přečtení znaku b, následováno posloupností znaků a, musí B dosáhnout akceptujícího stavu při načtení nejvíce m znaků a. Protože B má pouze konečné množství akceptujících stavů, musí některý z nich při běhu nad slovem $(ba^m)^\omega$ navštívit nekonečně často, tudíž B toto slovo přijímá. □

Uzávěrové vlastnosti

Při zkoumání operací nad jazyky, je nutné vědět, jaké jsou vlastnosti jazyka, který je výsledkem operace. Např. pokud provedeme průnik dvou jazyků, chceme vědět, jestli nově vzniklý jazyk patří do stejné třídy. Pokud operace nad dvěma jazyky určité třídy vytváří opět jazyk stejné třídy, říkáme, že je tato operace nad třídou uzavřená. Znalost uzávěrových vlastností ω-jazyků (které přijímají Büchiho automaty) je pro tuto práci velice důležitá. Pokud provedeme nějakou operaci nad ω-jazyky, chceme vědět, jestli je výsledkem opět ω-jazyk.

Uzavřenost vůči sjednocení, průniku a konkatenaci s regulárním jazykem je ukázána v důkazu lemmatu 2.2. Zbývá nám tedy ukázat, že jazyky, které Büchiho automaty přijímají, jsou uzavřeny vůči doplňku.

Tvrzení 2.4. *Jazyky rozpoznávané Büchiho automaty, jsou uzavřené vůči doplňku. Jinými slovy, máme-li Büchiho automat B, který přijímá jazyk $L(B)$, můžeme sestrojit Büchiho automat B', který rozpoznává jazyk $\Sigma \setminus L(B)$.*

Důkaz. Důkaz, který je zde popsán, byl uveden samotným Büchim v [3]. Následuje tedy jeho konstrukce. Mějme Büchiho automat $B = (Q, \Sigma, \delta, S, F)$. Dále je použito značení $q \underset{w}{\rightarrow} q'$ a obdobně značení $q \overset{F}{\underset{w}{\rightarrow}} q'$ pro takové dva stavy, kdy platí, že $q \underset{w}{\rightarrow} q'$ a běh z q do q' obsahuje akceptující stav. Mějme relaci ekvivalence $\sim_B$ na množině Σ^+ takovou, že pro každé $v, w \in \Sigma^+, v \sim_B w$ právě tehdy, když pro všechny $q, q' \in Q$ platí

$$q \underset{w}{\rightarrow} q' \Leftrightarrow q \underset{v}{\rightarrow} q'$$

a zároveň platí, že

$$q \xrightarrow[w]{F} q' \Leftrightarrow q \xrightarrow[v]{F} q'.$$

Podle definice, každá funkce $f : Q \to 2^Q \times 2^Q$ definuje třídu relace $\sim_B$. Takovou třídu označíme jako L_f. Funkce f bude vypadat následovně. $w \in L_f$ právě tehdy, když $\forall p \in Q$ a $(Q_1, Q_2) = f(p)$ platí $\forall q_1 \in Q_1 : p \xrightarrow[w]{} q_1 \wedge \forall q_2 \in Q_2 : p \xrightarrow[w]{F} q_2$. Zmiňme, že $Q_2 \subseteq Q_1$. Následující tři teorémy poskytují návod na konstrukci komplementu automatu B za použití tříd ekvivalence $\sim_B$.

Tvrzení 2.5. *$\sim_B$ má konečně mnoho tříd ekvivalence a každá z těchto tříd je regulární jazyk.*

Důkaz. Poněvadž existuje pouze konečný počet funkcí $f : Q \to 2^Q \times 2^Q$, má relace ekvivalence $\sim_B$ pouze konečný počet tříd. Nyní ukážeme, že L_f je regulární jazyk. Pro každou dvojici $p, q \in Q$ a $i \in \{0, 1\}$ vytvoříme nedeterministický konečný automat $A_{i,p,q} = (\{0,1\} \times Q, \Sigma, \delta_1 \cup \delta_2, \{(0,p)\}, \{(i,q)\})$, kde $\delta_1 = \{((0,q_1), a, (0,q_2)) \mid (q_1, a, q_2) \in \delta\} \cup \{((1,q_1), a, (1,q_2)) \mid (q_1, a, q_2) \in \delta\}$ a $\delta_2 = \{((0,q_1), a, (1,q_2)) \mid q_1 \in F \wedge (q_1, a, q_2) \in \delta\}$. Nechť $Q' \subseteq Q$. $\alpha_{p,Q'} = \bigcap_{q \in Q'} L(A_{1,p,q})$ pak označuje množinu slov w, pro která platí $\forall q \in Q' : p \xrightarrow[w]{F} q$. Dále definujeme $\beta_{p,Q'} = \bigcap_{q \in Q'} (L(A_{0,p,q}) \setminus L(A_{1,p,q}) \setminus \epsilon)$ jako množinu neprázdných slov, která provedou přechod automatu B z p do všech stavů z Q' aniž by při tom navštívily některý z akceptujících stavů. Nakonec definujme $\gamma_{p,Q'} = \bigcap_{q \in Q'} (\Sigma^+ \setminus L(A_{0,p,q}))$, což je množina všech neprázdných slov w, pro které neplatí $\forall q \in Q' : p \xrightarrow[w]{} q$. Z těchto definicí pak plyne, že $L_f = \bigcap_{p \in Q} \{\alpha_{p,Q_2} \cap \beta_{p,Q_1 - Q_2} \cap \gamma_{p,Q-Q_1} \mid (Q_1, Q_2) = f(p)\}$. □

Tvrzení 2.6. *Pro každé slovo $w \in \Sigma^\omega$ existují třídy $\sim_B$ L_f a L_g takové, že $w \in L_f (L_g)^\omega$.*

Důkaz. Pro důkaz tohoto teorému je využita Ramseyho věta pro nekonečné grafy (z anglického *infinite Ramsey theorem*), která je dokázána v [14]. Nechť $w = a_0 a_1 \ldots$ a $w(i,j) = a_i \ldots a_{j-1}$. Uvažujme množinu přirozených čísel $\mathbb{N}$. Nechť jsou třídy ekvivalence $\sim_B$ barvy dvouprvkových podmnožin $\mathbb{N}$. Přiřadíme barvy následovně. Pro každé $i < j$ nechť barva $\{i, j\}$ je ekvivalenční třída, ve které se vyskytuje $w(i,j)$. Díky Ramseyho větě můžeme najít nekonečnou množinu $X \subseteq \mathbb{N}$ takovou, že každá dvouprvková podmnožina X má stejnou barvu. Nechť $0 < i_1 < i_2 < \ldots \in X$. Nechť f je definující funkce takové ekvivalenční třídy, že $w(0, i_0) \in L_f$. Dále nechť g je definující funkce takové ekvivalenční třídy, že $\forall j > 0, w(i_{j-1}, i_j) \in L_g$. Tím dostáváme, že $w \in L_f (L_g)^\omega$. □

Tvrzení 2.7. *Nechť jsou L_f a L_g ekvivalenční třídy $\sim_B$. $L_f (L_g)^\omega$ je buď podmnožina $L(B)$ nebo disjunktní s $L(B)$.*

Důkaz. Předpokládejme slovo $w \in L(B) \cap L_f (L_g)^\omega$. V ostatních případech je důkaz snadný. Nechť r je akceptující běh B nad vstupem w. Musíme nyní ukázat, že každé slovo $w' \in L_f (L_g)^\omega$ patří také do $L(B)$, tzn. že existuje akceptující běh B nad slovem w'. Protože $w \in L_f (L_g)^\omega$, mějme $w = w_0 w_1 w_2 \ldots$ takové, že $w_0 \in L_f$ a $\forall i > 0, w_i \in L_g$. Nechť q_i je stav v r, do kterého se dostaneme po přečtení $w_0 \ldots w_i$. Pak nechť I je množina indexů takových, že $i \in I$ právě tehdy, když část běhu od stavu q_i do stavu q_{i+1} obsahuje stav z F. I tedy musí být nekonečná množina. Obdobně rozdělme slovo w'. Nechť $w' = w'_0 w'_1 w'_2 \ldots$ je slovo takové, že $w'_0 \in L_f$ a $\forall i > 0, w'_i \in L_g$. Sestrojíme r' indukčně a to následovně. Nechť první stav r' je stejný jako v r. Podle definice L_f nyní můžeme zvolit segment běhu nad slovem w'_0, abychom dosáhli stavu q_0. Podle indukčního předpokladu máme běh nad slovem

$w'_0 \dots w'_i$, který dosáhl stavu q_i. Podle definice L_g můžeme r' rozšířit o segment běhu, který čte w'_{i+1} tak, že dosáhne stavu s_{i+1} a navštíví při tom stav z F, pokud $i \in I$. Takto získané r' obsahuje nekonečně mnoho segmentů, ve kterých se vyskytuje stav z F, poněvadž I je nekonečná. Z toho plyne, že r' je akceptující běh a tedy $w' \in L(B)$. □

Z předchozích tvrzení plyne, že můžeme reprezentovat jazyk $\Sigma^\omega \setminus L(B)$ jako konečné sjednocení ω-jazyků ve formě $L_f(L_g)^\omega$, kde L_f a L_g jsou ekvivalenční třídy $\sim_B$. A tedy, že $\Sigma^\omega \setminus L(B)$ je ω-jazyk, který lze převést na Büchiho automat. □

Kapitola 3

Algoritmy pro Büchiho automaty

Pro potřeby této kapitoly mějme dva Büchiho automaty definované jako $B_1 = (Q_1, \Sigma_1, \delta_1, S_1, F_1)$ a $B_2 = (Q_2, \Sigma_2, \delta, S_2, F_2)$, které přijímají jazyky $L(B_1)$ a $L(B_2)$.

3.1 Pomocné algoritmy

Některé algoritmy pro svůj běh požadují, aby Büchiho automat měl pouze jeden počáteční stav. Všechny Büchiho automaty lze bez ztráty obecnosti na tento převést. Algoritmus pro konstrukci takového automatu je uveden jako algoritmus 1.

Algoritmus 1 Konstrukce Büchiho automatu B s jedním počátečním stavem, který přijímá jazyk $L(B) = L(B_1)$.

Vstup: B_1
Výstup: $B = (Q, \Sigma, \delta, s_0, F)$, který přijímá jazyk $L(B_1)$

- $Q \leftarrow \{q \mid q \in Q_1 \wedge (q \in S_1 \Rightarrow (\exists q' \in Q_1, a \in \Sigma : (q', a, q) \in \delta_1))\} \cup \{s_0\}$
- $\Sigma \leftarrow \Sigma_1$
- $\delta \leftarrow \{(q, a, q') \mid q, q' \in Q, a \in \Sigma : (q, a, q') \in \delta_1\} \cup \{(s_0, a, q) \mid a \in \Sigma,$ $q \in Q_1, (s, a, q) \in \delta_1$ pro nějaké $s \in S_1\}$
- $F \leftarrow F_1$

Algoritmus pro výpočet komplementárního automatu k danému Büchiho automatu požaduje, aby vstupní automat měl totální přechodovou funkci. Aby přechodová funkce byla totální, musí z každého stavu q být definovaný přechod (q, a, q') pro všechny symboly a vstupní abecedy. Algoritmus 2 je konstrukcí Büchiho automatu s totální přechodovou funkcí.

3.2 Sjednocení

Konstrukce Büchiho automatu B, který přijímá jazyk $L(B) = L(B_1) \cup L(B_2)$ již byla uvedena při důkazu uzavřenosti ω-jazyků nad sjednocením. Tato konstrukce je uvedena v podobě algoritmu (vizte algoritmus 3).

Algoritmus 2 Konstrukce Büchiho automatu B s totální přechodovou funkcí, který přijímá jazyk $L(B) = L(B_1)$.

Vstup: B_1
Výstup: $B = (Q, \Sigma, \delta, s, F)$ takový, že δ je totální přechodová funkce, a který přijímá jazyk $L(B_1)$

- $Q \leftarrow Q_1 \cup \{q_n\}$
- $\Sigma \leftarrow \Sigma_1$
- $\delta \leftarrow \delta_1 \cup \{(q, a, q_n) \mid a \in \Sigma, q \in Q_1 : \forall q' \in Q_1, (q, a, q') \notin \delta_1\}$
- $S \leftarrow S_1$
- $F \leftarrow F_1$

Algoritmus 3 Konstrukce Büchiho automatu přijímajícího jazyk $L(B_1) \cup L(B_2)$.

Vstup: B_1 a B_2
Výstup: $B = (Q, \Sigma, \delta, S, F)$, pro který $L(B) = L(B_1) \cup L(B_2)$

- $Q \leftarrow Q_1 \cup Q_2$
- $\Sigma \leftarrow \Sigma_1 \cup \Sigma_2$
- $\delta \leftarrow \delta_1 \cup \delta_2$
- $S \leftarrow S_1 \cup S_2$
- $F \leftarrow F_1 \cup F_2$

3.3 Průnik

Jak vytvořit Büchiho automat, který by přijímal průnik dvou ω-jazyků jsme si ukázali již při důkazu uzavřenosti těchto jazyků nad průnikem. Nyní představím o něco efektivnější algoritmus jeho tvorby. Je založen na myšlence, že v původním automatu se stavy se 2 v třetí komponentě vyskytují pouze jako koncové stavy, do kterých vstoupíme a okamžitě v dalším přechodu se vracíme do stavů s třetí komponentou rovnou 0. Místo, abychom při průchodu stavem, jehož druhá komponenta je akceptující stav, zvýšili třetí komponentu o jedna, vrátíme se zpět do stavu, kdy čekáme na akceptující stav prvního automatu (tedy do stavu s třetí komponentou rovnou 0). Jako akceptující stavy pak můžeme označit ty, které mají jako druhou komponentu některý z akceptujících stavů druhého automatu a třetí komponentu mají rovnou 1. Zřejmě pokud máme běh, který prochází nekonečně často akceptujícím stavem v první komponentě (simulující první automat) a zároveň pokud prochází nekonečně často akceptujícím stavem ve druhé komponentě (simulující běh druhého automatu), bude tento běh akceptující i v automatu takto zkonstruovaném. Algoritmus 4 uvádí tuto konstrukci [12].

Při některých aplikacích používající Büchiho automaty je běžné, že se v nich vyskytne automat, který má všechny své stavy přijímající. Takový automat dostaneme například převodem z Kripkeho struktury. Kripkeho struktura se běžně používá pro modelování systému (více v podkapitole 3.6). Pro takové případy, kdy jeden z automatů, ze kterých chceme vytvořit průnik, má všechny své stavy přijímající, můžeme algoritmus ještě vylepšit. Není totiž třeba čekat až bude první komponenta přijímající stav. Lze tedy vynechat třetí komponentu úplně. Výsledná konstrukce je uvedena jako algoritmus 5. [5]

Algoritmus 4 Konstrukce Büchiho automatu přijímajícího jazyk $L(B_1) \cap L(B_2)$.

Vstup: B_1 a B_2
Výstup: $B = (Q, \Sigma, \delta, S, F)$, pro který $L(B) = L(B_1) \cap L(B_2)$

- $Q \leftarrow Q_1 \times Q_2 \times \{0, 1\}$
- $\Sigma \leftarrow \Sigma_1 \cap \Sigma_2$
- $\delta \leftarrow ((q_1, q_2, x), a, (q_1', q_2', y))$, kde
  - $q_1, q_1' \in Q_1$, $q_2, q_2' \in Q_2$, $a \in \Sigma$
  - $(q_1, a, q_1') \in \delta_1$ a $(q_2, a, q_2') \in \delta_2$
  - Třetí komponenty jsou vypočítány následovně.
    - $q_1 \in F_1$ a $x = 0$, pak $y = 1$
    - $q_2 \in F_2$ a $x = 1$, pak $y = 0$
    - Jinak $x = y$
- $S \leftarrow S_1 \times S_2 \times \{0\}$
- $F \leftarrow Q_1 \times F_2 \times \{1\}$

Algoritmus 5 Konstrukce Büchiho automatu přijímajícího jazyk $L(B_1) \cap L(B_2)$, kde automat B_1 má všechny stavy akceptující.

Vstup: B_1 a B_2, $Q_1 = F_1$
Výstup: $B = (Q, \Sigma, \delta, S, F)$, pro který $L(B) = L(B_1) \cap L(B_2)$

- $Q \leftarrow Q_1 \times Q_2$
- $\Sigma \leftarrow \Sigma_1 \cap \Sigma_2$
- $\delta \leftarrow ((q_1, q_2), a, (q_1', q_2')) \in \delta$ právě tehdy, když $(q_1, a, q_1') \in \delta_1$ a $(q_2, a, q_2') \in \delta_2$
- $S \leftarrow S_1 \times S_2$
- $F \leftarrow Q_1 \times F_2$

3.4 Doplněk

Konstrukce použita v důkazu věty 2.4 je zajímavá z pohledu historického. Ovšem z implementačního hlediska je velmi neefektivní. Automat vytvořen Büchiho algoritmem má $2^{2^{O(n)}}$ stavů, kde n je počet stavů původního automatu [8]. V této kapitole je uvedena konstrukce, tak jak ji vytvořili Hrishikesh Karmarkar a Supratik Chakraborty v článku [8]. Postup, který je uveden je optimalizací algoritmu Friedguta, Kupfermana a Vardiho uvedeném v [6], konkrétně varianty uvedené Schewem v [17]. Konstrukce je založena na minimálním lichém ohodnocení DAG-běhů nad slovy, která přijímá komplement nedeterministického Büchiho automatu (tuto myšlenku prvně představili Kupferman a Vardi v [11]).

Algoritmus 6 předpokládá, že vstupní Büchiho automat má pouze jeden počáteční stav. Zároveň musí mít totální přechodovou funkci. Algoritmy pro převod automatu na takový, který splňuje uvedené podmínky, jsou představeny v podkapitole 3.1. Dále při popisu konstrukce doplňku je použita přechodová funkce ve tvaru $\delta : (Q \times \Sigma) \to 2^Q$ a přechod (q, a, q') značen jako $q' \in \delta(q, a)$. Pro zjednodušení zápisu budiž $\delta(Q', a) = \bigcup_{q \in Q'} \delta(q, a)$, kde $Q' \subseteq Q$.

3.4.1 Základní pojmy a struktury

Mějme nedeterministický Büchiho automat $B = (Q, \Sigma, \delta, s_0, F)$, kde funkce δ je totální (neboli je definována na celém svém definičním oboru).

Budiž $w \in \Sigma^\omega$ ω-slovo přijímané automatem B. Pak G_w je orientovaný acyklický graf (dále jen DAG z anglického *directed acyclic graph*), který reprezentuje všechny možné běhy automatu B nad slovem w. Graf $G_w = (V, E)$, kde $V \subseteq Q \times \mathbb{N}_0$ a $E \subseteq V \times V$. Kořen stromu, který G_w tvoří, je $(s_0, 0)$. Dále pro všechna $i > 0$ platí $(q, i) \in V$ právě tehdy, když existuje běh ρ automatu B nad slovem w takový, že $\rho(i) = q$. Pro E platí, že $((q, i), (q', j)) \in E$, právě tehdy, když $(q, i), (q', j) \in V, j = i+1, q' \in \delta(q, w(i))$. Graf G_w nazýváme DAG-během automatu B nad slovem w. Vrchol $(q, l) \in V$ budeme nazývat:

- F-vrchol, pokud $q \in F$.
- F-volný, pokud v grafu G_w neexistuje žádný F-vrchol, který by byl dosažitelný z (q, l).
- Konečný, pokud je počet z něj dosažitelných vrcholů konečný.

Dále, pro všechna $l \geq 0$, množina $\{(q, l) \mid (q, l) \in V\}$ tvoří úroveň l grafu G_w. Graf G_w nazveme akceptující, pokud v něm existuje nekonečná cesta $\rho_{G_w} = (s_0, 0)(q_1, 1)(q_2, 2)\ldots$, která obsahuje nekonečně mnoho F-vrcholů $q_i \in F$, $i \in \mathbb{N}$. V opačném případě je graf neakceptující.

Označme množinu $\{1, 2, \ldots, k\}$ jako $[k]$ a $[k]^{odd}$ jako množinu všech lichých čísel z $[k]$. Mějme automat B s n stavy a slovo $w \in \Sigma^\omega$. Dále mějme graf G_w, který je DAG-běh automatu B nad slovem w. Zavedeme ohodnocení $r : V \to [2n]$, které splňuje následující požadavky:

- $\forall (q, l) \in V : r((q, l)) \in [2n]^{odd} \Rightarrow q \notin F$,
- $\forall ((q, l), (q', l + 1)) \in E : r((q', l + 1)) \leq r((q, l))$.

Řekneme, že r je liché, pokud je každá nekonečná cesta v G_w někdy zachycena v lichém ohodnocení. Jinak jej nazveme sudým ohodnocením. $max\_odd(r)$ je pak největší lichá hodnota, kterou r nabývá.

Dalším krokem je zavedení tzv. ohodnocení stupně (z anglického *level ranking*). Ohodnocení stupně automatu B je funkce $g : Q \to [2n] \cup \{\bot\}$. Na tuto funkci klademe podmínku.

Pokud je $g(q)$ lichý, pak $q \notin F$. Budiž $\mathcal{L}$ množina všech ohodnocení stupňů automatu B. Mějme $g, g' \in \mathcal{L}, Q' \subseteq Q, a \in \Sigma$, pak g' pokrývá (g, Q', a), pokud $\forall q \in Q', \forall q' \in \delta(q, a)$: $g(q) \neq \perp \Rightarrow (g'(q') \neq \perp \wedge g'(q') \leq g(q))$. Význam $max\_odd(g)$ je obdobný, jako u ohodnocovací funkce z předchozího odstavce.

Ohodnocení r grafu G_w indukuje ohodnocení stupňů pro každý stupeň $l \geq 0$ grafu G_w. Mějme množinu $Q_l = \{q_l \mid (q, l) \in V\}$, která označuje množinu stavů na úrovni l. Pak ohodnocení stupně g indukované r na úrovni l je následující:

- $g(q) = r((q, l))$ pro všechna $q \in Q_l$,
- $g(q) = \perp$ jinak.

Pokud q a q' jsou ohodnocení stupňů indukovaná r pro stupně l a $l + 1$, pak g' pokrývá $(g, Q_l, w(l))$. Ohodnocení stupně je těsné (z anglického **tight**), pokud platí, že nejvyšší ohodnocení v rozsahu g je liché a zároveň, pro všechna $i \in [max\_odd(g)]^{odd}$ existuje stav $q \in Q$ pro který platí, že $g(q) = i$.

Tvrzení 3.1. *Následující tvrzení jsou ekvivalentní:*

(i) Všechny cesty v G_w vidí pouze konečný počet F-vrcholů.

(ii) Existuje liché ohodnocení G_w.

Důkaz sestrojením konstrukce pro $(i) \Rightarrow (ii)$ je ukázán v [11]. Mějme automat B s n stavy, ω-slovo $w \in \overline{L(B)}$ a graf G_w, který je DAG-během automatu B nad slovem w. Důkaz z [11] pak induktivně vytváří nekonečnou řadu orientovaných acyklických grafů $G_0 \supseteq G_1 \supseteq \ldots$, kde pro všechna $i \geq 0$ platí:

(i) $G_0 = G_w$,

(ii) $G_{2i+1} = G_{2i} \setminus \{(q, l) \mid (q, l)$ je konečný v $G_{2i}\}$,

(iii) $G_{2i+2} = G_{2i+1} \setminus \{(q, l) \mid (q, l)$ je F-volný v $G_{2i+1}\}$.

Nyní můžeme definovat ohodnocení $r^{KV}_{B,w}$ grafu G_w následovně. Pro každý vrchol $(q, l) \in G_w$ $r^{KV}_{B,w}((q, l)) = 2i$ pokud (q, l) je konečný v G_{2i} a $r^{KV}_{B,w}((q, l)) = 2i + 1$ pokud je (q, l) F-volný v G_{2i+1}. Kupferman a Vardi ukázali, že $r^{KV}_{B,w}$ je liché ohodnocení [11]. V [8] je pak dokázáno, že takto vytvořená funkce je minimální (neboli neexistuje ohodnocení, které by bylo pro kterýkoli vrchol menší). Důkaz pro následující tvrzení lze najít v [6].

Tvrzení 3.2. *Následující tvrzení jsou ekvivalentní:*

1. *Všechny cesty grafu G_w obsahují pouze konečně mnoho F-vrcholů.*
2. *Existuje liché ohodnocení G_w.*

3.4.2 Konstrukce

Automat vytvořený pomocí konstrukce z algoritmu 6 má tu vlastnost, že pokud přijme slovo $w \in \Sigma^\omega$, pak přiřadí ohodnocení r grafu G_w, které je v každém vrcholu stejné jako $r^{KV}_{B,w}$. Toho je docíleno tím, že se nedeterministicky napodobuje přiřazování ohodnocení, které je použito k výpočtu $r^{KV}_{B,w}$ (vizte konec předchozí podkapitoly). Bližší popis omezení stavů a přechodů, která umožňují, aby automat přijímající $L(B_1)$ splňoval předchozí požadavek, jsou popsány v [8].

Stav automatu z konstrukce uvedené v algoritmu 6, značený jako (P, O, g), postupně sleduje ohodnocení vrcholů DAG-běhu na aktuálním stupni. P je množina stavů, které jsou dosažitelné poté, co jsme přečetli konečný prefix slova. g je těsné ohodnocení stupně. Množina O pak slouží ke kontroluje, jestli pro každý vrchol z O ohodnocený k existuje nějaká cesta, která vede do stavu ohodnoceného $k-1$. Pokud je k sudé, kontroluje též, jestli všechny cesty vedoucí z takového vrcholu vedou do nějakého stavu ohodnoceného $k-1$. Pokud existuje cesta pro všechny vrcholy O ohodnocené dvojkou do stavu ohodnoceného jedničkou, vytvoříme O znova. Do nově vytvořené množiny O pak dáme všechny stavy, které jsou ohodnoceny největším lichým číslem z nynějšího rozsahu ohodnocení. Celý proces kontroly je pak započat od začátku. Popis konstrukce je uveden jako algoritmus 6.

Algoritmus 6 Konstrukce Büchiho automatu B přijímajícího jazyk $L(B) = \Sigma^\omega \setminus L(B_1)$ podle [8].

Vstup: B_1 s n stavy, dále předpokládejme, že $S = \{s_0\}$ a že δ_1 je totální
Výstup: $B = (Q, \Sigma, \delta, S, F)$, pro který $L(B) = \Sigma^\omega \setminus L(B_1)$

$Q \leftarrow \{(P, O, g) \mid P \subseteq Q_1, O \subseteq P, g \in \mathcal{L} : P \neq \emptyset$ a buď $O = \emptyset$
nebo $\exists k \in [2n-1] \forall q \in O, g(q) = k\}$
$S \leftarrow \bigcup_{i \in [2n-1]} \{(P, O, g) \mid P = \{s_0\}, g(s_0) = i, O = \emptyset\}$
$\Sigma \leftarrow \Sigma_1$
$\forall a \in \Sigma, (P', O', g') \in \delta((P, O, g), a)$, pokud jsou splněny následující podmínky.

1. $P' = \delta_1(P, a)$, g' pokrývá (g, P, a).
2. $\forall q \in P \setminus F$ existuje $q' \in \delta_1(q, a)$ takové, že $g'(q') = g(q)$.
3. $\forall q \in P \cap F$ platí jedna z následujících podmínek.
   (a) Existuje $q' \in \delta_1(q, a)$ takové, že $g'(q') = g(q)$.
   (b) Existuje $q' \in \delta_1(q, a)$ takové, že $g'(q') = g(q) - 1$.
4. Navíc platí následující omezení.
   (a) Pokud g není těsné ohodnocení stupně, pak $O = O'$.
   (b) Pokud g je těsné ohodnocení stupně a $O \neq \emptyset$, pak
      i. Pokud všechna $q \in O$ je $g(q) = k$, kde k je sudé, pak
         - Budiž $O_1 = \delta_1(O, a) \setminus \{q \mid g'(q) < k\}$.
         - Pokud $O_1 = \emptyset$, pak $O' = \{q \mid q \in Q_1 \wedge g'(q) = k - 1\}$, jinak $O' = O_1$.
      ii. Pokud všechna $q \in O$ je $g(q) = k$, kde k je liché, pak
         - Pokud $k = 1$, pak $O' = \emptyset$.
         - Pokud je $k > 1$, pak budiž $O_2 = O \setminus \{q \mid \exists q' \in \delta_1(q, a), (g'(q') = k - 1)\}$.
           – Pokud $O_2 \neq \emptyset$, pak $O' \subseteq \delta_1(O_2, a)$ taková, že $\forall q \in O_2 \; \exists q' \in O'$, $q' \in \delta_1(q, a) \wedge g'(q') = k$.
           – Pokud $O_2 = \emptyset$, pak $O' = \{q \mid g'(q) = k - 1\}$.
   (c) Pokud g je těsné ohodnocení stupně a $O = \emptyset$, pak $O' = \{q \mid g'(q) = max\_odd(g')\}$.

$F \leftarrow \{(P, O, g) \mid$ g je těsné ohodnocení stupně a $O = \emptyset\}$

3.5 Kontrola prázdnosti

Kontrola prázdnosti je problém, který je pro Büchiho automaty rozhodnutelný (důkaz je uveden například v [18]). Předpokládejme, že jazyk přijímaný Büchiho automatem B je neprázdný. Pak ovšem existuje běh ρ automatu B, který prochází nekonečně často některým akceptujícím stavem. Existuje tedy takový sufix ρ' běhu ρ, který všechny stavy, jež obsahuje, prochází nekonečněkrát. Z každého stavu ρ' se tedy můžeme dostat do všech ostatních stavů tohoto sufixu. To znamená, že stavy ρ' jsou součástí nějaké silně souvislé komponenty grafu reprezentujícího automat B. Zároveň tato silně souvislá komponenta musí obsahovat některý z akceptujících stavů a musí být dostupné z některého z počátečních stavů.

Postup pro nalezení akceptujícího běhu automatu B je tedy následující. Nejdříve musíme najít cestu z počátečního stavu do některého akceptujícího stavu. Pokud taková cesta existuje, zjistíme, jestli je tento stav součástí nějaké silně souvislé komponenty. Pokud tomu tak je, našli jsme běh automatu, který je akceptující. Pokud zjistíme, že žádný přijímající stav není dostupný nebo že žádný z dostupných není součástí silně souvislé komponenty, můžeme prohlásit, že je jazyk prázdný.

Základem zde uvedeného algoritmu je Tarjanův algoritmus prohledávání do hloubky (dále pouze DFS z *anglického depth first search*) pro hledání silně souvislých komponent. Použita je konkrétně jeho varianta uvedená v [5]. Algoritmus tvoří dvě do sebe vnořené DFS procedury. První procedura se snaží nalézt akceptující stav dostupný z některého stavu. Pokud jej nalezne zavolá druhou proceduru, která se snaží nalézt cyklus, který začíná a končí v tomto stavu. Pokud je takový cyklus nalezen, je algoritmus ukončen a vrácena hodnota signalizující neprázdnost jazyka. V opačném případě se řízení vrací první proceduře. Pokud není nalezen žádný akceptující stav, který je součástí cyklu, ukončíme algoritmus. Jazyk je pak prázdný. Tento algoritmus lze ještě optimalizovat. V druhé DFS proceduře můžeme značit stavy, které jsme již vyzkoušeli. U těchto stavů nemá smysl znovu hledat, jestli jsou součástí cyklu. Pokud jsou na nějakém cyklu, pak jej objevíme při prvním průchodu druhým DFS a algoritmus bude ukončen. Pokud jsme na stav narazili podruhé, pak je zřejmé, že není součást žádného cyklu a nemusíme jej brát již v potaz.

Algoritmus je možné upravit tak, aby v případě neprázdnosti jazyka vracel příklad přijímaného slova. Toto slovo je složeno z předpony a nekonečně se opakující přípony. Předponu tvoří stavy, které prošla první DFS procedura od počátečního stavu k akceptujícímu stavu. Přípona je pak určena stavy na smyčce nalezené druhou DFS procedurou.

Algoritmus 7 popisuje výše uvedený algoritmus. Ten pracuje s frontou, nad kterou jsou definovány tři funkce. První se stará o uložení do fronty (*enqueue*), druhá o vybrání prvky z fronty (*dequeue*) a poslední z nich kontroluje, jestli je argument ve frontě (*enqueued*).

3.6 Převod z Kripkeho struktury

Jedno z možných využití knihovny, kterou má tato práce za úkol vytvořit, je pro Model checking. Model checking slouží k formální verifikaci systémů. Jedna z možností, jak tyto systémy a jejich chování modelovat, je pomocí Kripkeho struktury. Mějme množinu základních pozorování AP (zkratka z anglického *atomic proposition*).

Definice 3.1. *Kripkeho struktura M nad AP je čtveřice $M = (Q, S, R, L)$, kde:*

- *Q je množina stavů,*
- *$S \subseteq Q$ je množina počátečních stavů,*

Algoritmus 7 Kontrola prázdnosti jazyka.

Vstup: B_1

Výstup: $True$ pokud je jazyk prázdný a $False$ pokud obsahuje aspoň jedno slovo

```
global f1 ← {}
global f2 ← {}

function EMPTINESS
    for all s ∈ S1 do
        dfs1(s)
    exit(True)

function DFS1(q)
    f1 ← f1 ∪ {q}
    enqueue(q)
    for all q' ∈ {qs | w ∈ Σ, (q, w, qs) ∈ δ1} do
        if q' ∉ f1 then
            dfs1(q')
    if q ∈ F1 then
        dfs2(q)
    dequeue(q)

function DFS2(q)
    f2 ← f2 ∪ {q}
    for all q' ∈ {qs | w ∈ Σ, (q, w, qs) ∈ δ1} do
        if enqueued(q') then
            exit(False)
        if q' ∉ f2 then
            dfs2(q')
```

- $R \subseteq Q \times Q$ *je přechodová funkce, která je totální. To znamená, že pro každý stav* $q \in Q$ *existuje stav* $q' \in Q$ *takový, že* $(q, q') \in R$,
- $L : Q \to 2^{AP}$ *je funkce, která přiřazuje stavům množinu atomických pozorování, která platí v daném stavu.*

Na první pohled je definice Kripkeho struktury podobná Büchiho automatům. Dokonce Kripkeho struktura přímo koresponduje s Büchiho automatem, který má všechny stavy akceptující. Množina chování systému M je stejná jako jazyk $L(B)$ takového Büchiho automatu. Převod M na B je popsán v algoritmu 8.

Algoritmus 8 Převod Kripkeho struktury na Büchiho automat.

Vstup: Kripkeho struktura nad AP $M = (Q, S, R, L)$
Výstup: $B = (Q', \Sigma, \delta, S', F)$

- $Q' \leftarrow Q \cup \{s_0\}$
- $\Sigma \leftarrow 2^{AP}$
- $\delta \leftarrow \{(q, \alpha, q') \mid q, q' \in Q, \alpha = L(q') \wedge (q, q') \in R\} \cup \{(s_0, \alpha, q) \mid q \in S \wedge \alpha = L(q)\}$
- $S' \leftarrow \{s_0\}$
- $F \leftarrow Q'$

Kapitola 4

Návrh knihovny

4.1 Očekávané použití

Výsledným produktem této práce bude knihovna, která bude poskytovat rozhraní pro práci s Büchiho automaty. Bude umožňovat jejich načtení ze souboru, případně opětovné uložení na disk. Práce s automaty bude zahrnovat základní operace, kterými jsou sjednocení, průnik a doplněk. Další funkcí, kterou bude knihovna implementovat, je zjištění, jestli je jazyk přijímaný automatem prázdný. V opačném případě bude poskytnut příklad slova, které testovaný Büchiho automat přijímá.

Pro účely testování a jako názorná ukázka funkčnosti knihovny bude vytvořen program, který bude výslednou knihovnu používat ke své práci. Tento program bude spouštěn z příkazové řádky, aby mohl být součástí skriptů, případně složitějších příkazů. Možnost použití programu ve skriptech usnadní testování na velkém objemu ukázkových příkladů.

Lze očekávat, že knihovna (případně ukázkový program) budou používány na různých operačních systémech, proto by výsledná implementace měla být pokud možno přenositelná. Pro jazyk, který bude zvolen jako implementační by tedy měl existovat překladač (případně interpret) pro většinu běžně dostupných systémů. Jedním z takových jazyků je například Python [16].

4.2 Datové struktury

Pro práci s Büchiho automaty je potřeba tyto uložit tak, aby se s nimi dobře pracovalo v algoritmech, které budou implementovány. Tvorbu automatu, lze rozdělit na několik úrovní. Nejnižší míru abstrakce představují symboly abecedy a stavy automatu. Ty jsou dále nedílné. O úroveň výš je potřeba říct, jak budu ukládat jednotlivé části automatu. Konkrétně seznamy stavů Q, S a F a abeceda Σ. Složitější než předchozí a též důležitější z pohledu práce algoritmů, je konstrukce přechodové funkce. Další úroveň představuje samotný Büchiho automat. V kapitole 3 je pak uveden algoritmus pro převod z Kripkeho strukturu. Knihovna tedy bude muset umět uložit i tu.

Co se týče použitých struktur, rozhodl jsem se pro větší prostorovou složitost. Přičemž věřím, že ušetřím nějaký čas při práci algoritmů. Pokud při implementaci nenarazím na nepřekonatelnou překážku, rozhodl jsem se použít jazyk Python. Tumu jsem též podřídil návrh knihovny a struktur, které bude používat pro uložení automatů.

Na nejnižším stupni abstrakce jsou abeceda a stav automatu. Tyto základní stavební kameny nebudou nikde ukládány, budou pouze součástí struktur na vyšších úrovních abs-

trakce. Každý stav, či symbol abecedy bude zastoupen svým identifikátorem. Tím může být například řetězec či n-tice tvořena jinými identifikátory. Musí se ovšem jednat o datový typ, který je statický neboli neměnný. Stejně budou modelovány stavy Kripkeho struktury. Symboly abecedy Kripkeho struktury, tedy množiny základních pozorování, budou modelovány jako neměnná (statická) množina řetězců označujících jednotlivá atomická pozorování.

Další úrovní abstrakce jsou množiny Σ, O, S a F. Ty budou ukládány množiny. Množinu představuje struktura obsahující jednotlivé unikátní prvky, nad kterou jsou definovány alespoň operace průnik, sjednocení a vyhledání prvku. Stejně budou modelovány množiny Q a S Kripkeho struktury.

Část návrhu nejvíce náchylná k neefektivitě (ať už z hlediska prostoru či použití v algoritmech) je určení struktury pro přechodovou funkci. Rozhodl jsem se modelovat tuto funkci jako dvě úrovně slovníků. Pro názornost mějme $(q, a, Q) = \{(q, a, q') \mid (q, a, q') \in \delta\}$. Klíčem slovníku na první úrovni bude identifikátor stavu, ze kterého vede přechod (tedy stav q). Hodnotou tohoto slovníku bude další slovník. Klíčem vnořeného slovníku bude symbol abecedy, tedy a. Hodnotou tohoto slovníku pak bude množina Q, která představuje všechny stavy, do nichž se dá z q přejít načtením písmene a. Pro tuto variantu jsem se rozhodl, protože je často potřeba hledat přechody ze stavu q přes nějaký znak a. Při této implementaci by nalezení takových přechodů mělo být snadné. Pro Kripkeho strukturu je situace méně složitá. Pro její množinu R bude použit seznam dvojic (q, q'). Pro množinu L pak bude použit slovník, který bude simulovat funkci $L : Q \to 2^{AP}$.

Büchiho automat bude tvořen objektem. Ten bude obsahovat množiny Q, Σ, S, F a slovník δ jako atributy. Dále bude implementovat jednotlivé algoritmy pro práci nad Büchiho automaty. Tyto algoritmy lze rozdělit na tři skupiny. Ty které poskytují nějaké informace o instanci, funkce měnící atributy instance a ty, které vytváří novou instanci, která je výsledkem některé operace nad Büchiho automaty. Algoritmy, které byly představeny, případně jsou potřeba:

- informativní - kontrola prázdnosti přijímaného jazyka, příklad přijímaného slova, kontrola jestli $Q = F$, test na správnost zadání automatu, přístup k jednotlivým atributům,
- transformační - sjednocení vstupních stavů, vytvoření totální funkce, změna jednotlivých atributů,
- operace nad Büchiho automaty - sjednocení, průnik, doplněk.

Objekt, který bude tvořit Kripkeho strukturu bude obsahovat následující atributy a metody. Atributy tvořící: abecedu (množinu základních pozorování), množinau Q a S a dále množinu dvojic R a slovník L. Jediné metody Kripkeho struktury budou test na správnost zadání a metody přistupující k jednotlivým atributům struktury.

Knihovna bude umožňovat načtení a opětovné uložení automatů z a do souboru. Tomuto účelu budou sloužit dvě samostatné funkce. Formát souborů, se kterými bude knihovna pracovat je uveden v kapitole 4.3

4.2.1 Varianty a rozšíření

Objektu modelujícímu Büchiho automat může být případně přidán atribut určující typ akceptační podmínky. Ten bude využit v případě rozšíření knihovny o práci s různými akceptačními podmínkami (Rabinova, Mullerova, Streettova apod.). Automaty s jinou akceptační podmínkou budou sloužit pouze pro účely načtení, případně uložení do souboru.

Navíc je bude možné převádět na nedeterministický Büchiho automat a zpět. Samotné algoritmy budou pracovat pouze s nedeterministickými Büchiho automaty.

Existuje více možných struktur, které lze využít pro implementaci Büchiho automatů. Ukážu tedy druhou variantu. Tato varianta je méně náročná na prostor, avšak při implementaci v jazyce Python by mohla působit velice nepřehledně. Následuje krátký popis této varianty. Stav je objekt tvořený atributy: identifikátor a seznam následovníků. Identifikátor je řetězec jednoznačně určující daný stav v rámci jednoho Büchiho automatu (je potřeba především při načítání a ukládání). Seznam následovníků je slovník, kde klíčem je symbol abecedy a hodnotou množina stavů, do kterých vede přechodová funkce při načtení symbolu z klíče. Případné dva další atributy by uchovávaly informaci, jestli je stav akceptační, případně vstupní. Büchiho automat by pak obsahoval pouze množinu všech jeho stavů a abecedu, se kterou pracuje.

4.3 Formát vstupu

Na formát pro ukládání a načítání automatů (respektive Kripkeho struktury) touto knihovnou jsou kladeny následující nároky. Vstupní soubor musí být dobře čitelný jak pro člověka, tak pro knihovnu. Také by měl obsahovat co nejméně redundantních informací. V každém souboru bude zadán právě jeden automat. Nejdříve musí být určeno, která struktura je v souboru zapsána (například jaká varianta automatu). Podle toho budou pak vypadat následující řádky souboru. Popis vstupního souboru bude nyní představen po částech.

První řádek souboru bude mít následující tvar.

```
<struktura> [<typ>]
```

Kde <*struktura*> může nabývat těchto hodnot: `Buchi`, pokud soubor obsahuje Büchiho automat, nebo `Kripke` v případě, že soubor popisuje Kripkeho strukturu. Abychom mohli programu říci, kterou variantu automatu soubor obsahuje, může být uveden nepovinný parametr <*typ*>. Tento parametr je zde pro případ, kdyby jsem se rozhodl rozšířit knihovnu o práci s variantami Büchiho automatů (například Mullerův automat).

Obsah dalších řádků záleží na struktuře, kterou chceme definovat. Pro dvě základní varianty jsou požadovány tyto vstupy:

- Pro Büchiho automat - seznam stavů, abeceda, přechodová funkce.
- Pro Kripkeho strukturu - seznam stavů, abeceda (základní pozorování), relace přechodů, ohodnocovací funkce.

Jednotlivé typy vstupů nyní popíši.

Řádky pro definování stavů vypadají následovně.

```
states
   <stav> <stav> <stav> ...
```

Část definující stavy systému začíná neodsazeným slovem *states*. Na dalších řádcích pak následuje seznam stavů systému, odsazené od začátku řádku bílým znakem. Jednotlivé stavy jsou zadávány jako řetězce, které je jednoznačně identifikují. Jediná omezení na jméno jsou, aby neobsahovalo bílé znaky, nezačínalo symbolem `*` a nekončilo znakem `$`. Počáteční stavy označíme symbolem `*` na začátku identifikátoru. Akceptující stavy pak znakem `$` na konci identifikátoru.

Abecedu definujeme obdobně jako stavy. Začátek sekce je značen slovem *alphabet*. Následují řádky obsahující jednotlivé symboly abecedy. Ty musí být opět odsazené. Znaky abecedy opět zadáváme jako řetězce, které je jednoznačně identifikují (v rámci abecedy).

Řádky popisující přechodovou funkci vypadají takto.

```
delta
   <stav> <symbol abecedy> <stav> <stav> ...
```

Klíčové slovo *delta* opět označuje začátek sekce. Na každém řádku (který je opět odsazený bílým znakem) je uvedena množina přechodů (q, a, Q'). První stav na řádku udává q. Poté následuje načtený znak a. Nakonec je uveden seznam stavů, do kterých lze přejít z q při načtení a.

Relace přechodů Kripkeho struktury je zapsána obdobně jako přechodová funkce Büchiho automatu. Nejdříve je uveden výchozí stav a poté všechny stavy, do kterých z něj lze přejít. Začátek sekce je označen slovem *transitions*.

```
transitions
   <stav> <stav> <stav> ...
```

Konečně ohodnocovací funkce se zapisuje obdobně jako přechodová funkce Büchiho automatu. Sekce začíná slovem *labels*. Každý následující řádek pak obsahuje ohodnocení jednoho stavu. Nejdříve je uveden ohodnocovaný stav, dále pak seznam všech základních pozorování, které v tomto stavu platí.

```
labels
   <stav> <symbol abecedy> <symbol abecedy> <symbol abecedy> ...
```

Pro lepší čitelnost, může soubor obsahovat komentované řádky. Pokud řádek začíná na symbol `#`, pak se celý řádek nebude nikterak interpretovat.

Na konec této podkapitoly je uveden příklad zapsání Büchiho automatu do souboru. Mějme Büchiho automat
$B = (\{s_0, q_1\}, \{a, b\}, \{(s_0, a, s_0), (s_0, b, s_0), (s_0, a, q_1), (q_1, a, q_1)\}, \{s_0\}, \{q_1\})$. Tento automat lze zapsat následovně:

```
Buchi
states
#počáteční stav
   *s_0
#koncový stav
    q_1$
alphabet
   a b
delta
# (q,  a,  Q)
   s_0 a   s_0 q_1
   s_0 b   s_0
   q_1 a   q_1
```

Kapitola 5

Implementace

5.1 Třída `Buchi`

Třída `Buchi` modeluje Büchiho automat. Uchovává kompletní informace o automatu a implementuje metody pro práci s nimi. Třída obsahuje tyto základní atributy:

- `Q` - množina všech stavů automatu,
- `S` - množina všech počátečních stavů,
- `F` - množina všech akceptujících stavů,
- `alphabet` - množina symbolů abecedy,
- `delta` - ukládá přechodovou funkci δ

Množiny `Q`, `S`, `F` a `alphabet` jsou implementovány typem `set` (množina). Tento typ pro ukládání prvků využívá hashovací funkci, což umožňuje rychlé vyhledávání prvků. Tato vlastnost je výhodná zejména ve chvíli, kdy se nad těmito množinami provádí operace jako průnik či sjednocení. Přechodová funkce je ukládána v grafové podobě. Pro její implementaci je použit *slovník*. Klíči jsou výchozí stavy (vrcholy grafu), hodnotami jsou hrany, které z tohoto stavu vychází. Tyto hrany jsou implementovány též jako slovníky. Symboly abecedy jsou jejich klíči a množiny stavů, do kterých lze přejít z výchozího stavu při načtení symbolu, jsou jejich hodnotami. V jazyce Python by tedy zápis přechodů ze stavu q při načtení symbolu a do stavů $q1$ a $q2$ vypadal takto:

```
{'q': {'a': set(['q1', 'q2']), ...}, ...}.
```

Knihovna si též zajišťuje, že v rámci celého programu jsou stavy pojmenovány unikátně. Toto je zajištěno třídní proměnnou `next_state`. Tato proměnná slouží jako čítač. Při inicializaci nového automatu se každý název stavu přeloží na číslo obsažené v proměnné `next_state`, která se následně inkrementuje o jedna. Použití tohoto mechanismu má jednu nevýhodu. Tou je ztráta původního názvu stavu. Při implementaci jsem se ovšem zaměřil na funkčnost jednotlivých algoritmů. Některé algoritmy totiž vyžadují, aby množiny stavů byly disjunktní. Toto je tímto přístupem zajištěno implicitně, bez nutnosti měnit názvy stavů v programu používající tuto knihovnu.

Třída `Buchi` dále obsahuje několik pomocných atributů sloužících k uchování stavu výpočtu prázdnosti jazyka. Jsou to atributy `__f1`, `__f2`, `__queue`, `__queue2` a `empty`. Atribut `empty` obsahuje informaci o prázdnosti jazyka přijímaného automatem, případně hodnotu

`None`, pokud ještě prázdnost nebyla ověřena. Informace o prázdnosti je ukládána, protože ji využívá více metod třídy a její výpočet může být v některých případech náročný. Ostatní pomocné atributy jsou použity v metodě `is_empty` k výpočtu. Tyto proměnné jsou zachovány pro použití v metodách `reduce_Q` a `word_example`.

5.1.1 Metody

Metody implementované v třídě `Buchi` se dělí do tří kategorií. První kategorií jsou metody, které poskytují informace o automatu. Do této kategorie patří metody `is_empty`, `test_is_empty`, `is_delta_total` a `word_example`. Druhou kategorii tvoří metody měnící některé základní atributy. Metody spadající do této kategorie jsou `total_delta`, `pom_init_state` a `redukce_bez_ekvivalence`. Metoda `is_empty` též mění některé atributy instance, ale pouze jako vedlejší efekt, navíc nemění automat samotný. V poslední kategorii jsou metody, které jako výsledek svého běhu produkují nový automat. Tyto metody implementují základní operace průnik (metoda `prunik`), sjednocení (metoda `sjednoceni`) a doplněk (metoda `doplnek`). Do třetí kategorie též patří metoda `copy`, která vytváří kopii automatu. Dále třída `Büchi` obsahuje pomocné metody. Ty jsou popsány u metod, ke kterým patří.

V dalších podkapitolách jsou popsány implementace jednotlivých metod se zaměřením na zajímavé části řešení. Kromě výše uvedených metod v této části jsou uvedeny i standardní metody. Konstruktor třídy `__init__` s pomocnou metodou `nacti` a metodu na převod do řetězcové podoby `__str__`.

Metoda __init__ (Q, S, F, delta, alphabet, soubor)

Konstruktor třídy `Buchi` přijímá šest nepovinných parametrů. Prvních pět představuje jednotlivé části Büchiho automatu. Argumenty `Q`, `S`, `F` a `alphabet` jsou množiny představující stavy a abecedu automatu. Parametr `delta` je slovník v podobě, v jaké je popsán na začátku kapitoly 5.1. Parametr `soubor` obsahuje jméno vstupního souboru, ze kterého má být načten automat. Pro inicializaci nového automatu lze použit buď název souboru obsahující Büchiho automat nebo prvních pět parametrů pro zadání automatu přímo z programu. Pokud je zadáno všech šest parametrů, je použit automat ze souboru.

O načítání automatu ze souboru se stará pomocná metoda `nacti`. Jediným jejím parametrem je název souboru, který obsahuje popis Büchiho automatu. Jak vypadá takový soubor je popsáno v kapitole 4.3. Při načítání jsou použity regulární výrazy rozpoznávající jednotlivé části souboru. Je nutné, aby řádky definující samotné prvky automatu (tzn. stavy, symboly abecedy, přechody) byly od začátku řádku odsazeny bílým znakem. Algoritmus se může nacházet v několika stavech. Ten je uložen v proměnné `cast` a nabývá hodnot od 0 do 3. Ve stavu 0 se hledá označení automatu (v případě této třídy řádek začínající slovem `Buchi`). Po nalezení tohoto řádku se algoritmus přenese do stavu 1. V tomto stavu hledá definici sekce (`states`, `alphabet` nebo `delta`). Po jejím nalezení se stav mění na 2. Zde se čeká na řádek obsahující jednotlivé prvky (stavy, symboly apod.). Po nalezení se stav mění na 3. Zde se načítají další řádky s definicemi prvků. Pokud se zde narazí na definici sekce, je stav změněn na 2 a začíná načítání další sekce. Je-li v kterémkoli stavu nalezen jiný vstup (vyjma komentáře a prázdného řádku), je program ukončen výjimkou.

Samotné načítání pak pracuje takto. Postupně se prochází řádky vstupního souboru a pomocí regulárních výrazů se zjišťuje, co řádek obsahuje. Pokud řádek začíná symbolem `#`, případně je prázdný (obsahuje pouze bílé znaky), je vynechán. U ostatních částí je

vždy nejdříve zkontrolován stav a pokud odpovídá načtenému řádku, provedou se příslušné akce. Při prohledávání se ukládají názvy sekcí, jejichž definice byly nalezeny. Pokud po ukončení čtení souboru nebyly nalezeny všechny sekce, je vyvolána výjimka a program je ukončen. Pokud vše proběhlo v pořádku jsou navráceny struktury obsahující jednotlivé části automatu.

Po případném načtení dat ze souboru jsou naplněny hlavní atributy instance. Názvy stavů jsou přeloženy na unikátní čísla pomocí třídní proměnné `next_state`. Jednotlivé symboly abecedy se převedou na řetězce. Toto je opatření pro případ, že by symboly byly definovány jako datový typ, z něhož nelze vypočíst hash (jako např. seznam, množina, slovník). Nakonec jsou vytvořeny i zbylé množiny.

Metoda __str__ ()

Metoda `__str__` je další standardní metodou, kterou obsahuje většina tříd v Pythonu. Tato metoda je volána, když je potřeba řetězcová reprezentace. Slouží tedy k převodu instance třídy na její řetězcovou podobu (např. funkce `str`). Pomocí této metody tedy lze vypsat automat v textové podobě (ať už na standardní výstup či do souboru). Výstupem je řetězec, který odpovídá specifikaci uvedené v kapitole 4.3. Tedy soubor naplněný takto vypsaným řetězcem lze opakovaně načíst. Kromě nutných částí obsahuje též výpis několik komentářů, které usnadňují čtení automatu ze souboru (pro člověka).

Metoda is_empty (S, smazat)

Metod má dva nepovinné argumenty `S` a `smazat`. Pokud tyto argumenty nejsou předány, pak tato metoda vrací informaci o prázdnosti jazyka přijímaného automatem dané instance. Pokud je jazyk prázdný, vrací `True`, jinak vrací `False`s. Metoda též nastavuje atribut `empty`. Při opakovaném volání této metody se vrací pouze hodnota tohoto atributu. Tento mechanismus lze použít, protože třída `Buchi` neobsahuje metody, které by měnily automat dané instance tak, aby se změnil jazyk. Pokud by se programátor, vyžívající tuto knihovnu rozhodl změnit automat (například změnou přechodové funkce), bylo by nutné též změnit hodnotu `empty` na `None`. Argumenty `S` a `smazat` znamenají, že funkce `is_empty` je volána k jinému účelu. Její funkci využívá metoda `reduce` ke zjištění, které stavy jsou v automatu zbytečné a které naopak vedou k cíli.

Metoda `empty` využívá dvě pomocné metody - `__dfs1` a `__dfs2`. Každá z nich implementuje prohledávání grafu automatu do hloubky. První z nich hledá akceptující vrchol (vždy až při návratu) a druhá se pak snaží najít cyklus začínající a končící v tomto vrcholu. Jak celé toto hledání pracuje, je popsáno v algoritmu 7 na str. 20. Jako globální proměnné f_1 a f_2 slouží atributy `__f1` a `__f2`. Ty jsou implementovány jako množiny, což urychluje zjišťování členství prvků. V případě prohledávání velkých prázdných automatů, se výpočet urychlí. Dále jsou použity atributy `__queue` a `__queue2` k ukládání cesty od počátečního stavu k aktuálnímu vrcholu. Oproti algoritmu 7 je přidána druhá fronta, která se po ukončení používá k nalezení příkladu přijímaného slova.

Metoda test_is_empty (prunik)

Tato metoda přistupuje k hledání akceptujícího běhu z opačné strany než metoda `is_empty`. Nejdříve se pokusí zjistit, jestli existuje smyčka obsahující kterýkoli z akceptujících stavů. K hledání smyčky je použito prohledávání do hloubky metodou `__test_dfs2`. Tato metoda přijímá dva parametry. Prvním je aktuálně prohledávaný prvek, druhým je akceptu-

jící stav, do kterého se snažíme najít cestu. Metoda rekurzivně zkouší prohledat všechny potomky prohledávané stavy. Pokud je potomek hledaný akceptující stav, byla nalezena smyčka a můžeme se vrátit. Stavy na nalezené smyčce (byla-li nějaká nalezena) se přidá do množiny `__f_smycky`. Pokud není množina `__f_smycky` po prohledání všech akceptujících stavů prázdná, pokusí se metoda zjistit, jestli je některá z nalezených smyček dostupná z počátečních stavů. K tomuto účelu je použita metoda `__test_dfs1`. Pokud je dostupný některý stav z `__f_smycky`, pak není jazyk prázdný (existuje akceptující běh). V opačném případě je jazyk prázdný.

Tato metoda přijímá nepovinný argument `prunik`. Pokud byl automat vytvořen pomocí metody `prunik`, může být argument `prunik` nastaven na `True`. V tom případě bude výpočet ukončen okamžitě poté, co bude nalezena smyčka přes akceptující stav, protože víme, že automat obsahuje pouze dostupné stavy (což je vlastnost metody `prunik`).

Metoda is_delta_total ()

Metoda zjišťuje, zda je přechodová funkce totální. Tedy jestli z každého stavu vede přechod pro každý znak abecedy. Pokud tomu tak je, vrací `True`. V opačném případě vrací `False`. Metoda nejdříve zjistí, jestli je množina klíčů slovníku `delta` rovna množině všech stavů. Dále se zjišťuje, zda jsou definovány přechody z každého znaku, přes všechny znaky abecedy.

Metoda word_example ()

Tato metoda vrací příklad slova přijímaného automatem. Toto slovo vrací ve dvou seznamech. První seznam obsahuje prefix slova, tedy tu část, která je načtena před navštívením akceptujícího stavu, ze kterého vede dále smyčka. Druhý seznam pak obsahuje část slova, která se opakuje nekonečněkrát.

Metoda `word_example` nejdříve zjistí, zda je jazyk prázdný či nikoli. Toto zjištění má dvojí funkci. Pokud se zjistí, že je jazyk prázdný, vrátí se hned dva prázdné seznamy. V opačném případě metoda `is_empty` naplní atributy `__queue` a `__queue2`, které jsou dále použity pro nalezení slova. Jednotlivé symboly jsou počítány následovně. Fronta obsahuje stavy $[q_0 q_1 q_2, \ldots q_n]$. Ve *for* cyklu pro i od 1 do n se do výsledné množiny přidá znak, kterým se můžeme dostat ze stavu q_{i-1} do stavu q_i. K nalezení takového symbolu je použita pomocná funkce `__find_symbol_between`. Ta jako parametry bere dva stavy a vrací symbol, jehož načtením se můžeme dostat z prvního stavu do druhého. Znaky prefixu jsou vypočítány z fronty `__queue`. Druhá množina znaků je počítána ve dvou krocích. První projde frontu `__queue2`. Poté se zjistí, kolikátý je poslední stav této fronty v `__queue` a od tohoto místa se dopočítají znaky opět až k akceptujícímu stavu, kterým `__queue2` začíná.

Metoda total_delta ()

Tato metoda doplní přechodovou funkci tak, aby byla totální. Algoritmus 2 na str. 14 toto doplnění popisuje. Implementace algoritmu pracuje následovně. Nejdříve se vytvoří nový stav. Poté se dle algoritmu projdou všechny přechody. Pokud z nějakého stavu nelze přes některý symbol nikam přejít, doplní se přechod do nově vytvořeného stavu. Pokud byla funkce totální (a tedy žádný přechod nebyl doplněn) metoda končí. V opačném případě se nový stav přidá do množiny stavů (teprve nyní se zvýší hodnota `next_state`, která byla opět použita pro identifikaci nového stavu) a do přechodové funkce se přidají smyčky z nového stavu zpět přes všechny symboly abecedy.

Metoda pom_init_state ()

Metoda implementuje algoritmus 1 ze str. 13. Jejím účelem je upravit automat tak, aby měl pouze jeden vstupní stav. Na rozdíl od navrhovaného algoritmu v implementaci chybí odebírání nedostupných původních vstupních stavů. Kroky, které by tuto funkcionalitu zajišťovaly, by při velkých automatech byly příliš časově náročné a užitek by byl poměrně malý.

Metoda redukce_bez_ekvivalence ()

Tato metoda redukuje stavový prostor následujícím způsobem. Pokud bylo zjištěno dříve, že je jazyk přijímaný automatem prázdný, je automat přetvořen na takový, který má pouze jeden stav, který je zároveň počáteční, žádné akceptující stavy a přechodová funkce obsahuje pouze smyčky přes všechny symboly abecedy zpět do jediného stavu. V případě, že jazyk prázdný není, případně tato skutečnost nebyla prozatím zjišťována (tedy atribut `empty` obsahuje hodnotu `None`), provádí se samotná redukce.

Metoda používá tyto následující pomocné atributy. Atribut `__projite` obsahuje ty stavy automatu, které již byly prohledány algoritmem. Atribut `__red_f` je množina, jejíž prvky jsou stavy, které leží na nějakém přijímajícím běhu. Protože se stavy postupně prochází cyklem `for`, není možné automat měnit během výpočtu. Proto jsou stavy určené ke smazání nejdříve ukládány do proměnné `__smazat`, kde jsou uchovány až do chvíle, kdy mohou být tyto prvky smazány.

Samotný výpočet probíhá takto. Pro každý počáteční stav automatu, který prozatím algoritmus nenavštívil (není v proměnné `__projite`), se zavolá pomocná metoda `__red_clear_empty`. Tato rekurzivní metoda nejdříve přidá aktuálně procházený prvek (nazvěme ho q) do množiny již navštívených stavů. Poté nastaví pomocné proměnné metody `is_empty` tak, aby se provedl výpočet. V případě, že není známo jestli stav q leží na některém akceptujícím běhu (stavy z množiny `__red_f`), je zavolána metoda `is_empty` s parametrem `S` nastaveným na množinu obsahující stav q a parametrem `smazat` nastaveným na hodnotu `True`. Takto zavolaná metoda `is_empty` se chová, jako by nynější automat měl pouze jeden počáteční stav, a to q. Dále se při prohledávání vynechají stavy přidané do proměnné `__smazat` a prohledávání se úspěšně ukončí, je-li nalezen některý ze stavů v `__red_f`. Pokud je takový automat prázdný, jsou do množiny `__smazat` přidány všechny stavy, které metoda `is_empty` při svém běhu prošla. V opačném případě jsou stavy, které leží na nalezeném akceptujícím běhu, přidány do množiny `__red_f` a dále je metoda `__red_clear_empty` zavolána pro ty následovníky stavu q, které ještě nebyly prohledány a nejsou určeny ke smazání.

Po návratu do metody **redukce_bez_ekvivalence** jsou všechny stavy množiny `__smazat` z automatu odstraněny. Pokud stále existují počáteční stavy, které nebyly prohledány, spustí se pro jeden z nich další běh `__red_clear_empty`. Pokud již byly prohledány všechny počáteční stavy, jsou nakonec promazány nedostupné stavy (tedy ty, které nebyly prohledány). K odstranění stavů z automatu slouží další pomocná metoda `__odstran`. Ta bere jako parametr množinu stavů, které mají být odstraněny. Stavy jsou nejdříve odstraněny z jednotlivých množin (Q, S a F) a jsou též odstraněny přechody vycházející z těchto stavů. Poté jsou tyto stavy odebrány ze všech přechodů (postupným průchodem přechodovou funkcí).

Metoda prunik (B)

Metoda implementuje algoritmus 4 (vizte str. 15). Jediným vstupním argumentem (B) je instance třídy `Buchi`, s níž se má provést průnik jazyků. Výstupem je nová instance, která

modeluje Büchiho automat, jež přijímá jazyk $L(B_new) = L(B_self) \cap L(B)$. Automat se vytváří postupně. Nejdříve jsou vytvořeny všechny vstupní stavy nového automatu. Ty jsou následně přidány do zásobníku pro zpracování. Poté se postupně vybírají stavy ze zásobníku a hledají se přechody z nich vedoucí. Pokud přechod končí ve stavu, který ještě nebyl objeven (zatím se nevyskytuje v množině `Q`), je tento přidán do množiny stavů a zároveň do zásobníku ke zpracování. Pokud stav splňuje specifikaci akceptujícího stavu, je též přidán do množiny `F`. Tímto způsobem se tedy vytvoří pouze ty stavy, které jsou dostupné z počátečních stavů.

V průběhu výpočtu jsou jednotlivé stavy reprezentovány trojicí (datový typ *tuple*), kde jednotlivé prvky jsou: stav prvního automatu, stav druhého automatu a celé číslo, které slouží k zapamatování si, které akceptující stavy byly již navštíveny. Překlad této reprezentace na čísla je proveden v konstruktoru nového automatu, který je následně použit jako návratová hodnota.

Metoda sjednoceni (B)

Tato metoda implementuje algoritmus 3 na straně 14. Jako vstupní parametr B přijímá instanci třídy `Buchi`. Výstupem je nová instance třídy `Buchi`, která přijímá jazyk, jež je sjednocením jazyků argumentu a volající instance. Metoda pracuje přesně, jak je zapsáno v algoritmu 3. Vytvoří se nové množiny stavů sjednocením odpovídajících množin dvou vstupních automatů. Ty se následně použijí jako argumenty pro konstruktor nové instance.

Metoda doplnek ()

Nejrozsáhlejší a nejsložitější metodou je `doplnek`. Byla implementována dle algoritmu 6. Její stavy jsou, stejně jako u průniku, generovány postupně. Tím se v některých případech výrazně zmenší vygenerovaný stavový prostor, který je obecně u výpočtu doplňku velký problém. Algoritmus pro svůj běh potřebuje, aby automat, ke kterému je doplněk počítán, měl pouze jeden počáteční stav. Toto je nutné mít na paměti při tvorbě programu, který knihovnu využívá. Pokud tato podmínka není splněna, je vyvolána výjimka a běh je ukončen. Druhá podmínka pro běh algoritmu je totální přechodová funkce. Pokud přechodová funkce automatu není totální, je algoritmus opět ukončen výjimkou. Jednotlivé stavy jsou reprezentovány trojicí (datový typ *tuple*), kde první prvek je množina stavů P. Druhým prvkem je množina stavů O. Obě množiny jsou modelovány datovým typem množina (`set`). Posledním prvkem je funkce, ohodnocující prvky z množiny P (které představují stavy na některé úrovni). Tato funkce je reprezentována datový typem slovník (`dict`). Klíčem je vždy prvek z množiny P a hodnota je číslo z rozsahu 1 až $2 * n - 1$, kde n je počet stavů původního automatu. V této podobě jsou stavy reprezentovány pouze v zásobníku, pro zpracování (vizte dále v této kapitole). V samotných nově vytvářených množinách stavů (`Q`, `S`, `F`, `alphabet` a `delta`) jsou použity řetězcové reprezentace těchto trojic. Je to z toho důvodu, že datové typy množina a slovník využívají pro ukládání prvků (respektive klíčů) hashovací funkci. Ale protože trojice reprezentující stav obsahuje jak množinu, tak slovník (obě nelze použít pro hash), musí být tento stav převeden do řetězcové podoby a takto může být uložen do množiny, či být použit jako klíč slovníku.

Výpočet doplňku je komplexní problém, proto jeho je implementace popsána podrobněji. Nejdříve je třeba zjistit, jestli jazyk přijímaný automatem je prázdný. Pokud ano, je vytvořen automat o jednom stavu a přechody přes všechny symboly abecedy zpět do tohoto stavu. Tento jediný stav je zároveň počáteční i akceptující. Pokud jazyk přijímaný automatem není prázdný vytvoří se nejdříve množina počátečních stavů nového automatu.

Řetězcové reprezentace těchto stavů jsou uloženy do množin Q a S. Ve formě trojice jsou pak vloženy na zásobník nezpracovaných stavů (`new_q`).

Proces vytváření nových stavů poté pracuje do té doby, dokud je na zásobníku alespoň jeden prvek. V každém kroku se vybere ze zásobníku jeden stav. Pokud tento stav splňuje podmínky pro akceptující stav, je přidán do množiny F. Dále se vygenerují všechny jeho následníci dle pravidel uvedených v algoritmu 6. Vygenerované přechody se přidají do přechodové funkce. Pokud některý z nových stavů dosud nebyl nalezen, je přidán do množiny Q (v textové podobě) a je také přidán do zásobníku pro další zpracování.

Přechody z daného stavu přes každý znak jsou generovány následovně. Nejdříve je vypočtena nová množina P jako $P = \delta(P_{old}, a)$, kde a je symbol abecedy a P_{old} je vzata z právě prohledávaného stavu. Dalším krokem je vygenerování nové ohodnocovací funkce g. Těch je díky existenčnímu kvantifikátoru v podmínkách 2 a 3 algoritmu 6 několik. Pro každou vypočtenou ohodnocovací funkci je vypočteno nové O podle omezení uvedených v algoritmu 6. Existenční kvantifikátor se vyskytuje i v omezení 4. (b) ii. Tento krok tedy taky může generovat několik různých množin O. Nakonec je pro každou kombinaci P, O a g vytvořen stav. Je zkontrolováno, jestli se tento stav již nenachází v Q. V opačném případě se do této množiny přidá a zařadí se též do zásobníku ke zpracování. Vždy se vytvoří nový přechod.

Ke generování všech přípustných ohodnocovacích funkcí slouží pomocná metoda `__gen_level_rat`. Ta přijímá tři argumenty. Prvním je P_{old}, druhým je symbol abecedy a a třetím je ohodnocovací funkce právě zpracovávaného stavu. Výstupem je množina všech přípustných ohodnocovacích funkcí nové úrovně (kterou tvoří stavy v P). Důležitou vlastností je, že se generují pouze ty kombinace ohodnocení, které jsou přípustné podle omezení, která jsou kladena v algoritmu 6. Postup generování je následující. Nejdříve se vypočítají maxima, kterých může nová ohodnocovací funkce nabývat. Zde se uplatní první omezení, tedy že nová ohodnocovací funkce musí pokrývat trojici (g_{old}, P_{old}, a). Každý stav může nabývat nejvýše takové hodnoty, která je rovna nejnižšímu ohodnocení jeho předchůdců. Výpočet tohoto maxima provádí pomocná metoda `__find_maxims`, která má stejné parametry jako metoda `__gen_level_rat`.

Dále je zjištěno, kterých hodnot musí ohodnocovací funkce pro dané stavy nabývat. Zde se uplatňují podmínky 2 a 3. Pro každý stav z P_{old} jsou nalezeny všechny kombinace ohodnocení, které tyto podmínky splňují. Například, je-li neakceptující stav z P_{old} funkcí g_{old} ohodnocen číslem 4, musí alespoň jeden jeho potomek být též ohodnocen 4 (tedy můžou tak být ohodnoceny i dva a více). Je vytvořena množina, v níž jsou postupně ohodnoceny všechny kombinace potomků číslem 4 (ostatní prozatím zůstávají neohodnoceny). Pro generování těchto kombinací slouží metoda `__permut`. Jako vstupní parametry jsou jí předávány seznam doposud vytvořených kombinací, stav, který se bude k těmto kombinacím přidávat, hodnoty, kterých může nabývat a proměnná, ve které jsou uloženy použitelné hodnoty tohoto stavu. Je zde tedy brán zřetel na to, že daný stav pro jiného předchůdce může nabývat jen konkrétních hodnot a tedy jiné nemá smysl generovat. Výsledkem jsou všechny kombinace, vedoucí k přípustnému ohodnocení potomků jednoho stavu. Dalším krokem je spojení těchto ohodnocení tak, aby byly zároveň splněny podmínky pro všechny možné předchůdce. O připojování jednotlivých kombinací k sobě se stará pomocná metoda `__permut_nas`. Tato metoda přijímá čtyři parametry. Prvním je seznam již vygenerovaných ohodnocení. Druhý je slovník, obsahující informaci o tom, které stavy již byly nějak ohodnoceny. Třetí parametr jsou kombinace ohodnocení pro jednoho předka. Nakonec je ještě předána informace, které prvky jsou ohodnoceny v nově přidávaných kombinacích. Výsledkem je pak seznam kombinací po přidání ohodnocení splňující jednoho předka a informace

o již přidělených hodnotách. V tuto chvíli jsou tedy vytvořeny všechny kombinace ohodnocení, které splňují podmínky na ně kladené. Nakonec je tedy potřeba doplnit ohodnocení pro doposud neohodnocené stavy. Nejdříve jsou vypočítány hodnoty, které prozatím nebyly použity. Poté jsou všechny jejich kombinace přidány pomocí metody `__permut_final` tam, kde ohodnocení některého stavu zatím chybí. Nakonec tedy dostaneme seznam všech přípustných ohodnocení vrcholů.

Poslední část stavu, kterou je nutno vytvořit je množina O. Nejprve je třeba vědět, jestli je ohodnocení g_{old} těsné. K tomuto zjištění slouží metoda `__is_tight`. Ta bere jako parametr ohodnocovací funkci a vrací `True` v případě, že je těsná. Jinak vrací `False`. Všechna omezení na množinu O jsou jasně daná. Jediný problém je s omezením 4. (b) ii. Zde se nachází existenční kvantifikátor, což znamená, že možností, jak vytvořit množinu O je více. O generování všech přípustných množin, které splňují toto omezení se stará metoda `__kombinace`. Ta pracuje podobně jako metoda `__gen_level_rat`. Postupně vygeneruje podmnožiny, které splňují zadané omezení (pomocná funkce `__kombinace_pom`). Následně jsou k těmto podmnožinám přidány všechny kombinace ostatních stavů tak, aby se pokryly možné varianty (použita metoda `__kombinace_pom2`).

5.2 Třída `Kripke`

Druhou implementovanou třídou knihovny je `Kripke`. Jak již název napovídá, tato třída modeluje *Kripkeho strukturu*, která je definována v kapitole 3.6. Jedinými atributy třídy jsou:

- `Q` - množina všech stavů struktury,
- `S` - množina počátečních stavů,
- `alphabet` - množina jednotlivých atomických pozorování,
- `R` - relace reprezentující přechody mezi stavy,
- `L` - ohodnocovací funkce.

Stejně jako u třídy `Buchi` jsou i zde množiny `Q`, `S` a `alphabet` reprezentovány datovým typem *množina* (typ `set`). Relace `R` a ohodnocovací funkce `L` jsou ukládány jako *slovník* (typ `dict`). Klíčem ve slovníku reprezentujícího relaci `R` je stav, ze kterého přechod vychází. Hodnotou je pak *seznam* (typ `list`) stavů, do kterých lze přejít. Slovník ukládající funkci `L` má za klíče stavy struktury a jako hodnotu má podmnožinu atomických pozorování.

Kromě těchto základních atributů obsahuje třída ještě argument `next_state`. Ten, stejně jako u třídy `Buchi`, zajišťuje jednoznačnost jmen jednotlivých stavů. Jedná se o *třídní proměnnou*, která je pro všechny kripkeho struktury společná.

5.2.1 Metody

Třída `Kripke` má definovány pouze čtyři metody. První metodou je `__init__`. Jedná se o konstruktor. Metoda `nacti` slouží k načítání Kripkeho struktury ze souboru. Metoda `__str__` slouží pro převod struktury na její řetězcovou reprezentaci. Poslední metodou a nejdůležitější metodou je `to_buchi`. Ta slouží pro převod Kripkeho struktury na Büchiho automat. Jednotlivé metody jsou v krátkosti popsány níže.

Metoda __init__ (Q, S, R, L, alphabet, soubor)

Konstruktor třídy `Kripke` přijímá šest nepovinných parametrů. Při běžném použití se však předpokládá, že budou předány parametry pro pět základních atributů nebo jméno vstupního souboru s uloženou Kripkeho strukturou. V případě, že jsou předány všechny parametry, je výsledná třída naplněna daty ze vstupního souboru.

Metoda nacti (soubor)

Tato metoda pracuje stejně jako metoda `nacti` třídy `Buchi`. Podobu uložení kripkeho struktury popisuje kapitola 4.3. Aby byla definice struktury úplná, musí načítaný soubor obsahovat tyto sekce: `states`, `alphabet`, `transitions` a `labels`. Počáteční stavy jsou, stejně jako u Büchiho automatu, určeny hvězdičkou před jménem daného stavu.

Metoda __str__ ()

Metoda `__str__ ()` vytváří řetězcovou reprezentaci Kripkeho struktury. Řetězec je vytvořen tak, aby mohl být přímo uložen do souboru jako validní záznam Kripkeho struktury.

Metoda to_buchi ()

Tato metoda je implementací algoritmu 8 ze str. 21. Výstup této metody je instance třídy `Buchi`, která je převodem Kripkeho struktury na Büchiho automat. Nejdříve je vytvořen vstupní stav s_0 a je přidán ke stavům struktury. Poté je vytvořena abeceda výsledného büchiho automatu. Abecedou je potenční množina množiny všech základních pozorování. Množinu všech podmnožin vypočítává pomocná metoda `__potencni_mnozina`. Ta přijímá jako jediný parametr množinu. Výstupem je potenční množina vstupní množiny. Výpočet této množiny je proveden postupným přidáváním nových atomických pozorování do již vytvořených podmnožin. Z každé doposud vytvořené podmnožiny se vytvoří dvě nové. Jedna podmnožina obsahuje nově přidávané atomické pozorování, jedna nikoli. Dalším krokem je vytvoření přechodové funkce. Každý přechod z `R` tvoří jeden přechod budoucího automatu. Klíče slovníku `R` jsou výchozí stavy. Každý stav ze seznamu uloženém v hodnotě slovníku jsou pak cílové stavy. Symbol, který se při přechodu načítá je ohodnocení cílového stavu funkcí `L`. Do přechodové funkce `delta` jsou nakonec přidány přechody z nově vytvořeného počátečního stavu do všech počátečních stavů Kripkeho struktury. Nakonec je množina všech stavů zkopírována do množiny akceptujících stavů.

5.3 Pomocné skripty

Kromě vytvoření knihovny samotné obsahuje řešení několik pomocných skriptů. Tyto skripty využívají knihovnu a slouží pro testování knihovny, pro práci s Büchiho automaty z příkazové řádky a pro generování automatů. Tyto skripty jsou též součástí zadání a jsou v následující části popsány. Pokud skript vyžaduje načítání argumentů z příkazové řádky, je použita knihovna Pythonu `argparse`.

5.3.1 Knihovna pro generování náhodných automatů

Prvním pomocným skriptem je knihovna pro generování náhodných automatů `random_buchi`. Ta obsahuje dvě metody: `random_buchi` a `random_buchi_nd`. Obě generují náhodné Büchiho

automaty, každá však pracuje jinak a i výsledné automaty mají jiné parametry. Pro účely testování (obzvláště doplňku) je vhodná více druhá z nich, s přídomkem *nd*. Nyní je popsáno fungování obou těchto metod.

Pro práci s náhodností je použita standardní knihovna Pythonu `random`. Na začátku se inicializuje funkcí `seed`. Ta vyžaduje nějaké číslo pro inicializaci generátoru pseudonáhodných čísel. Pokud není žádné číslo předáno, je automaticky načteno z */dev/random* (pokud je dostupný), jinak je použit systémový čas. Pro generování reálných čísel od 0 do 1 je použita funkce `random`. Pro náhodný výběr prvku z množiny je použita funkce `choice`.

Metoda random_buchi (vel_Q, vel_F, vel_alph, hran, vel_S)

Tato metoda přijímá čtyři povinné parametry. Parametr `vel_Q` udává požadovaný počet stavů výsledného automatu, `vel_F` počet akceptujících stavů, `vel_alph` velikost abecedy. A nakonec `hran` je reálné číslo od 0 do 1, které udává, procento přechodů, které bude výsledný automat obsahovat. Parametr `vel_S` je nepovinný (jeho hodnota je přednastavena na 1) a udává počet počátečních stavů výsledného automatu.

Samotný automat se generuje následovně. Nejdříve je vytvořena množina stavů. Několik prvních stavů je vybráno jako počáteční stavy. Poté jsou z množiny stavů náhodně vybrány akceptující stavy. Nakonec je pro každou kombinaci stavů q_1, q_2 a symbolu abecedy a rozhodnuto, jestli stav q_2 patří do množiny $\delta(q_1, a)$. Pro každou kombinaci je vygenerováno náhodné číslo od 0 do 1. Pokud je toto číslo menší než hodnota `hran`, je přechod přidán do přechodové funkce. Nakonec je vrácen Büchiho automat s vypočítanými parametry.

Metoda random_buchi_nd (vel_Q, vel_F, vel_alph, hran, vel_S, d_hran)

Metoda přijímá stejné parametry jako `random_buchi`. Navíc je pouze nepovinný parametr `d_hran`, což je reálné číslo udávající, o kolik se má zmenšit pravděpodobnost přijetí dalšího přechodu z daného stavu.

Generování automatu probíhá podobně jako u `random_buchi`. Rozdíl je pouze v generování přechodové funkce. Pro každý stav a symbol abecedy je přechod do některého náhodně vybraného stavu přijat s pravděpodobností `hran`. Každý další přechod je přijat s pravděpodobností $hran * d\_hran$. Výhodou tohoto přístupu je, že takto generované automaty jsou téměř deterministické (význam *nd* v násvu je *nearly deterministic*). Počet nedeterministických přechodů je přímo úměrný parametru `d_hran`. Tato vlastnost se hodí například při počítání doplňku automatu, kde při velkém počtu nedeterministických přechodů počet stavů výsledného automatu narůstá velmi rychle.

5.3.2 Použití z příkazové řádky

Pro snadné použití knihovny z příkazové řádky slouží skript `use_buchi.py`. Program přijímá následující argumenty. Název akce, kterou má skript provést. Název vstupních a výstupních souborů podle akce. Navíc je volitelný argument `time`. Pokud je tento argument zadán, na konci běhu programu bude zobrazen čas trvání výpočtu. Akcemi, které umí program provádět, jsou jednotlivé operace nad automaty, jako například průnik a sjednocení dvou automatů, výpočet doplňku či zjištění, jestli je jazyk přijímaný automatem prázdný. Akce `gen_new` generuje náhodný automat pomocí metody `random_buchi_nd`. Počet vstupních souborů závisí na akci, kterou má program provést. Pokud požadovaná operace vyžaduje dva argumenty, musí být určeny oba dva. Všechny akce, s výjimkou `emptiness` potřebují ke svému běhu znát cílový soubor, kam bude zapsán výsledný automat. Výstupem akce

`emptiness` je pouze návratová hodnota. Dle zvyklostí *bashe* hodnota 0 říká, že zkoumaný automat přijímá prázdný jazyk. 1 znamená, že jazyk není prázdný. Při provádění akce `complementation` je načtený automat převeden do požadované formy (počet počátečních stavů se změní na jeden a doplní se přechodová funkce tak, aby byla totální).

5.3.3 Skripty k testování

Pro účely generování testovací sady automatů a následné testování knihovny na těchto příkladech jsou vytvořeny další skripty.

Testovací sadu automatů generuje skript `gen_autom.sh`. Tento bashový skript obsahuje příkazy pro vytvoření náhodných automatů o různých parametrech. Ručně je pak zapsán jeden prázdný automat a jeden automat přijímající jazyk L^ω. Ke generování několika náhodných automatů se stejnými parametry slouží skript `testovaci_BA.py`. Ten přijímá sedm povinných a jeden nepovinný parametr. Je mu předán počet automatů, které má vygenerovat. Následují jednotlivé (i nepovinné) parametry, které odpovídají parametrům funkce `random_buchi_nd`. Poslední parametr je nepovinný a určuje složku, do které budou automaty generovány.

Dalším krokem je testování nad vygenerovanými automaty. K tomuto účelu byl napsán skript `test.sh`. Ten nejdříve vyčistí složku s výsledky. Poté vygeneruje skript k samotnému provedení testů. Nakonec tento skript spustí. Ke generování testovacího skriptu slouží program `test.py`. Tento přijímá čtyři povinné argumenty. Prvním je akce, která se má provádět. Ta koresponduje s akcemi skriptu `use_buchi.py`. Dalším argumentem jsou soubory, na nichž se testy provádějí. Třetí je složka, kam se mají uložit výsledky. Posledním argument udává jméno výsledného skriptu. Program nejdříve zjistí jména vstupních souborů. Ty náhodně promíchá, aby se pokaždé testovala jiná kombinace automatů (to má význam především u binárních operací jako je průnik). Náhodné promíchání je zajištěno funkcí `shuffle` z knihovny `random`. Nyní jsou do výstupního souboru postupně zapisovány příkazy, využívající program `use_buchi.py`, zpracovávající jednotlivé soubory ze seznamu.

Kapitola 6

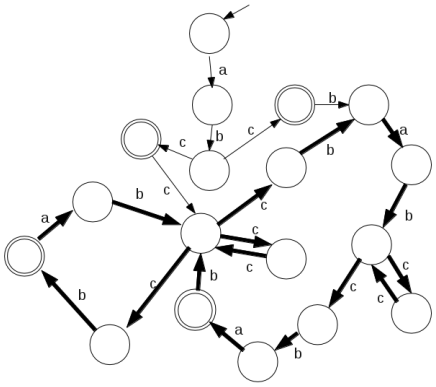
Výsledky testování

Testování bylo zaměřeno především na funkčnost knihovny. Pro účely testování bylo vygenerováno sto různých Büchiho automatů. Pro jejich tvorbu bylo využito skriptu `gen_autom.sh`. Ten generuje automaty s různě velkými počty stavů, akceptujících stavů, velikostmi abecedy a počtem přechodů. Navíc generuje jeden automat přijímající prázdný jazyk a jeden akceptující všechny řetězce jazyka Σ^ω. Testování probíhalo na notebooku s čtyřjádrovým procesorem s taktem 2.7 GHz, 8 GB paměti a 64 bitovým operačním systémem Fedora 17 (GNU/Linux). Na stroji byl nainstalován Python ve verzi 2.7.3. Soubory, které byly použity k testování, jsou uloženy na přiloženém médiu ve složce `testovaci_sada`.

Nad vygenerovanými automaty byly testovány veškeré funkce, které knihovna poskytuje. Nejdříve bylo zjišťováno, jestli se v kódu nevyskytují chyby, které by způsobovaly pád programu. Veškeré chyby, které se zpočátku vyskytovaly, byly opraveny a v průběhu dalšího testování se žádné pády či chyby nevyskytly. Při těchto testech byly též prováděny některé úpravy na algoritmech, kvůli zvýšení efektivity. Například redukce stavového prostoru původně pouze prováděla zjištění, zda je jazyk prázdný a v případě záporné odpovědi byly smazány ty stavy, které byly metodou `is_empty` prohledány, avšak nevedly k výsledku. Tato metoda sice v některých případech stavový prostor redukovala významně (třeba i o 75 %), avšak v jiných případech nebyl smazán ani jeden stav. Proto, aby bylo možné spouštět výpočet doplňku i na větších automatech byla vytvořena metoda `redukce_bez_ekvivalence`, která stavový prostor redukuje téměř na minimum (neprovádí test na ekvivalenci různých běhů).

Dále byly prováděny testy na funkčnost jednotlivých funkcí. Na menších automatech, kde byla možnost ověření správnosti ručně byla ověřena funkčnost jednotlivých metod. Největším problémem bylo zjištění správnosti algoritmu pro počítání doplňku. Většina automatů vytvořená touto funkcí je na ruční kontrolu příliš veliká. Výjimku tvoří automaty vzniklé z prázdných automatů a automatů přijímajících všechna slova. Avšak u automatů, které přijímají jednoduchý jazyk, je po redukci vzniklého automatu možné tento ručně projít a ověřit správnost. V některých případech je také možné provést doplněk doplňku a takto vzniklý automat redukovat a zkusit zjistit, jestli výsledný automat odpovídá původnímu. Příkladem automatu, pro který byl vypočítán dvojnásobný doplněk (každý doplněk byl redukován funkcí `redukce_bez_ekvivalence`) je např. automat z obrázku 2.1 na straně 6. Dvojitý doplněk je zobrazen na obrázku 6.1. Smyčky obsahující akceptující stavy jsou vyznačeny tučně.

Výpočet sjednocení a průniku nepřináší žádné potíže. Při výpočtu doplňku, případně při zjišťování prázdnosti jazyka ovšem knihovna naráží na problém velikosti automatů. Při velkých automatech (tzn. nad milion stavů) může trvat zjištění prázdnosti jazyka i něko-



Obrázek 6.1: Dvojitý doplněk automatu z obr. 2.1.

Název souboru	Výsledná velikost	čas výpočtu [s]
BA1_6_1_1_19_3.txt	427746	40
BA2_5_1_2_15_4.txt	345234	42
BA1_8_1_3_32_5.txt	3173164	569
BA0_25_1_2_56_3.txt	696	0.1
BA3_9_1_3_29_5.txt	154401	23

Tabulka 6.1: Tabulka obsahuje dobu výpočtu doplňku vybraných automatů a velikost výsledného automatu

lik hodin. Obzvlášť, pokud je jazyk prázdný, metoda musí prohledat celý stavový prostor, což může trvat neúměrně dlouho. Protože jedním z testů, které byly prováděny je test na prázdnost jazyka vytvořeného průnikem automatu s jeho doplňkem, kde se prochází velké automaty a výsledkem je prázdný jazyk, byla přidána do knihovny metoda `test_is_empty`. Tato metoda se nejdříve snaží zjistit, jestli existují smyčky obsahující některý akceptující stav. Pokud tomu tak je, pokusí se najít, jestli je taková smyčka dostupná. Pokud ne, je jazyk přijímaný automatem prázdný. Pokud ovšem víme, že testovaný automat byl vytvořen jako průnik dvou automatů pomocí metody `prunik`, můžeme ukončit výpočet s kladnou odpovědí okamžitě při zjištění, že existuje smyčka obsahující akceptující stav. Toto je způsobeno tím, že metoda `prunik` vytváří automat, kde jsou všechny stavy dostupné. Tedy existuje-li smyčka s akceptujícím stavem, je tato smyčka i dostupná z některého počátečního stavu.

V případě výpočtu doplňku je problém velikosti automatu ještě citelnější. Potíže se stavovou explozí přináší například vysoká míra nedeterminizmu. Tedy pokud se z jednoho stavu přes jeden symbol může automat přesunout do mnoha jiných stavů. V tomto případě je počet možných ohodnocení úrovně větší než $n * (n-1)^m$, kde n je počet stavů automatu a m je počet možných přechodů ze stavu přes jeden symbol. Obecně lze říci, že čím větší automat, případně čím větší počet přechodů a smyček, tím větší je výsledný automat a výpočet trvá déle. V tabulce 6.1 jsou uvedeny některé výsledky výpočtu doplňku při různých velikostech vstupního automatu.

Stavové explozi lze v některých případech předcházet redukcí automatu pomocí metody `redukce_bez_ekvivalence`. Obzvláště u automatů, které jsou doplňkem jiného automatu, může být rozdíl ve velikosti zásadní (viz. 6.2). Algoritmus pro výpočet doplňku totiž generuje velké množství stavů, které nejsou součástí žádného akceptujícího běhu a které tedy lze z automatu odstranit. V některých případech redukcí získáme automat, který sice má

Název souboru	Původní velikost [stavů]	Po redukci [stavů]	čas výpočtu [s]
BA1_6_1_1_19_3.txt	427746	4328	1897
BA2_5_1_2_15_4.txt	345234	78	981
BA3_6_1_1_22_4.txt	531576	822	4569

Tabulka 6.2: Tabulka ukazující změnu počtu stavů po redukci a čas výpočtu. Redukovány byly doplňky uvedených automatů.

velký počet stavů, ale i přesto výpočet jeho doplňku skončí v rozumném čase.

Kromě ruční kontroly, byl proveden test na prázdnost jazyku automatu, který je výsledkem operace průniku s jeho vlastním doplňkem. Tento test ovšem nepokrývá případ, kdy by automat vytvořený pomocí metody `doplnek` přijímal jazyk, který je pouze podmnožinou $\Sigma^\omega \setminus L(b)$. V nynějším stavu je to ovšem jediný test, kterým lze naznačit správnou funkčnost. Tímto testem prošly všechny automaty z testované množiny, pro které byl testovací stroj schopen vypočíst doplněk, průnik a následně prázdnost jazyka v omezeném čase (pro každou operaci 10 minut).

Při provádění testů nebyly zjištěny žádné chyby. Výsledky jednotlivých operací odpovídají očekávání. Potíže vyvstávají pouze při práci s většími automaty, kdy výpočet doplňku trvá dlouho, případně nedostačuje paměť počítače z důvodu stavové exploze. Toto je ovšem vlastnost samotného algoritmu pro výpočet doplňku a tudíž tento problém byl předvídán.

Kapitola 7

Závěr

Cílem této diplomové práce bylo vytvoření knihovny, která umožňuje práci s Büchiho automaty. Začátek práce je zaměřen na problematiku Büchiho automatů a pojmů, které jsou s nimi spjaty. V teoretické části (vizte kapitolu 2) jsou uvedeny základní pojmy teorie automatů. Tyto pojmy jsou nezbytné k pochopení zpracované problematiky. Problematika je popsána od samých základů, aby bylo možné pojmout význam a smysl jednotlivých jejích částí. Důležitou částí této teorie jsou *konečné automaty*. Těm je věnována velká část kapitoly 2. Je zde stručně uveden popis jejich chování a vlastností. Büchiho automaty, které patří mezi konečné automaty, jsou dále popsány detailněji se zaměřením na ty vlastnosti, které mají vliv na fungování knihovny (jako například uzávěrové vlastnosti).

Další část práce je zaměřena na návrh knihovny poskytující základní operace nad Büchiho automaty (vizte kapitoly 3 a 4). V první z kapitol zaměřených na návrh jsou postupně představeny algoritmy, které knihovna implementuje, včetně vysvětlení jejich fungování. V kapitole 4 je uveden návrh na implementaci knihovny. Jsou v ní uvedeny jednotlivé struktury, které jsou použity k modelování Büchiho automatů a je zde popsán návrh celého objektu. Na závěr kapitoly je uveden způsob, jakým budou generovány náhodné automaty pro účel testování a v jaké formě jsou automaty ukládány na disk.

Dalším krokem je implementace navržené knihovny a skriptů, které ji využívají. Při vytváření bylo postupováno dle návrhu. Jako programovací jazyk byl zvolen Python. Tento jazyk je dostatečně vysoko úrovňový, tudíž obsahuje velké množství pokročilých struktur, čehož bylo při implementaci využíváno. Je též aktivně vyvíjen a poskytuje dostatečný výkon, pro náročné operace nad automaty. Popis implementace jednotlivých objektů, které knihovna definuje, popis skriptů vytvořených pro práci s knihovnou a pro testování funkčnosti je uveden v kapitole 5. Hlavní část kapitoly je věnována popisu třídy `Buchi`, která modeluje Büchiho automaty. Popis je zaměřen na implementační detaily, kterými jsou řešeny jednotlivé algoritmy z kapitoly 3. Dále jsou popsány skripty, pomocí kterých je možné vytvářet automaty a pracovat s nimi z příkazové řádky.

Závěr práce je věnován testování funkčnosti knihovny (vizte kapitolu 6). Prvním krokem bylo vygenerování množiny automatů různých parametrů, na které poté bylo prováděno testování. V práci je uvedeno, jakým způsobem byla ověřena funkčnost a jakých poznatků bylo dosaženo. Pro testování byly vytvořeny pomocné skripty, jejichž účelem bylo zjednodušit a z části automatizovat samotné provádění testů. Kapitola se též zmiňuje o omezeních a překážkách při použití některých funkcí a popisuje možná východiska. Během testování byla též zjištěna potřeba některých pomocných metod, které byly následně implementovány do knihovny. Tyto nově vzniklé metody mají vliv na použitelnost a efektivitu knihovny pro některé konkrétní případy.

7.1 Rozšíření

Některá možná rozšíření práce jsou již navržena v kapitole 4.2.1. Některé aplikace pracující s Büchiho automaty využívají různých akceptačních podmínek. Jednou z těchto aplikací, která je významná z hlediska použití, je převod z LTL specifikace na zobecněný Büchiho automat [5]. Pomocí LTL specifikace lze popsat požadované chování systému, který má být formálně verifikován. Pokud by knihovna umožňovala převod z LTL formule na Büchiho automat (přes zobecněný Büchiho automat), bylo by možné ji pro verifikaci použít.

Knihovna nyní pro zjištění, jestli je jazyk automatu podmnožinou jiného, využívá vztahu $L(A) \subseteq L(B) \Leftrightarrow L(A) \cap L(\overline{B}) = \emptyset$. Dalším významným rozšířením by tedy mohl být efektivní algoritmus pro zjišťování inkluze jazyka. Mezi pokročilé a slibné metody patří simulací řízené testování na inkluzi jazyka založené na Ramseyho metodě [2].

Metoda, kterou nyní knihovna poskytuje pro redukci stavového prostoru, nedosahuje minima stavů a je neefektivní při práci s velkými automaty. Rozšířením knihovny o operaci minimalizace by zvýšilo její využití. Jako možnost se nabízí práce *Advanced automata minimization* [13].

Literatura

[1] Skripta do předmětu Základy teorie informace. `http://home.zcu.cz/~patrke/WWW-KMA/ZTI/ZTI-01tomaty.pdf` (cit. 2013/05), 2012.

[2] Abdulla, P.; Chen, Y.-F.; Clemente, L.; aj.: Simulation Subsumption in Ramsey-Based Büchi Automata Universality and Inclusion Testing. In *Computer Aided Verification*, *Lecture Notes in Computer Science*, ročník 6174, editace T. Touili; B. Cook; P. Jackson, Springer Berlin Heidelberg, 2010, ISBN 978-3-642-14294-9, s. 132–147, doi:10.1007/978-3-642-14295-6˙14. URL http://dx.doi.org/10.1007/978-3-642-14295-6_14

[3] Büchi, J. R.: On a decision method in restricted second order arithmetic. In *Proceedings of the International Congress on Logic, Methodology and Philosophy of Science*, Stanford University Press, 1960, s. 1–11.

[4] Chytil, M.: Automaty a gramatiky. 1984, ISBN 04-012-84.

[5] Edmund M. Clarce, J.; Grumberg, O.; Peled, D. A.: *Model checking*. MIT Press, 1999, ISBN 0-262-03270-8.

[6] Friedgut, E.; Kupferman, O.; Vardi, M.: Büchi Complementation Made Tighter. *International Journal of Foundations of Computer Science*, ročník 17, č. 04, 2006: s. 851–867.

[7] Habiballa, H.: *Teoretické základy informatiky I.* Ostravská univerzita v Ostravě, 2003, ISBN 80-7042-859-7.

[8] Karmarkar, H.; Chakraborty, S.: On Minimal Odd Rankings for Büchi Complementation. In *ATVA*, 2009, s. 228–243.

[9] Kocur, P.: Elektronická skripta do předmětu Teorie konečných automatů a formálních jazyků. `http://www.kvd.zcu.cz/cz/materialy/kafj/_kafj1.html` (cit. 2013/05), 2000.

[10] Kozen, D. C.: *Automata and computability*. Springer, 1997, ISBN 0-387-94907-0.

[11] Kupferman, O.; Vardi, M. Y.: Weak alternating automata are not that weak. In *ACM Transactions on Computational Logic*, ACM New York, 2001, s. 408 – 429.

[12] Křetínský, M.: Skripta do předmětu Automaty. `http://fi.muni.cz/usr/kretinsky/automaty.ps` (cit. 2013/05), 2002.

[13] Mayr, R.; Clemente, L.: Advanced automata minimization. In *Proceedings of the 40th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '13, New York, NY, USA: ACM, 2013, ISBN 978-1-4503-1832-7, s. 63–74, doi:10.1145/2429069.2429079.
URL http://doi.acm.org/10.1145/2429069.2429079

[14] Ramsey, F. P.: On a problem of formal logic. In *Proceedings of The London Mathematical Society*, 1930, s. 264–286.

[15] Rich, E.: *Automata, computability and complexity: theory and applications*. Pearson / Prentice Hall, 2008, ISBN 978-0-13-228806-4.

[16] van Rossum, G.; Fred L. Drake, J.: *The Python Language Reference Manual*. Network theory ltd, 2006, ISBN 0-954-16178-5.

[17] Schewe, S.: Büchi Complementation Made Tight. In *26th International Symposium on Theoretical Aspects of Computer Science*, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2009, ISBN 978-3-939897-09-5, s. 661–672.

[18] Thomas, W.: Automata on Infinite Objects. In *Handbook of Theoretical Computer Science*, MIT Press, 1990, s. 135–191.

[19] Černá, I.; Křetínský, M.; Kučera, A.: Skripta do předmětu Automaty a formální jazyky I.
http://is.muni.cz/elportal/estud/fi/js06/ib005/Formalni_jazyky_a_automaty_I.pdf
(cit. 2013/05), 2002.

[20] Češka, M.: Skripta do předmětu Teoretická informatika.
http://www.fit.vutbr.cz/study/courses/TI1/public/Texty/ti.pdf
(cit. 2013/05), 2002.