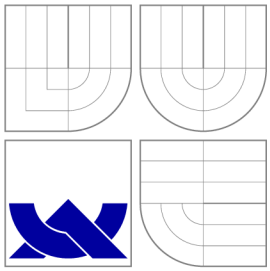
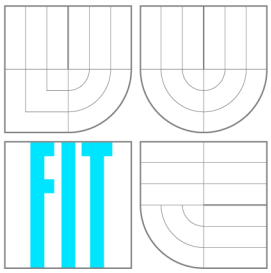


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

SYMBOLICKÁ REPREZENTACE KONEČNÝCH AUTOMATŮ

SYMBOLIC REPRESENTATION OF FINITE AUTOMATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JIŘÍ CHROMEČKA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ONDŘEJ LENGÁL

BRNO 2014

Abstrakt

V oblasti formální analýzy se často setkáváme s konečnými automaty s velkým množstvím stavů nad velkými abecedami. Jejich explicitní reprezentace může vyústit ve stavovou explozi a tento problém může být vyřešen použitím symbolické reprezentace schopné pracovat s celou množinou stavů najednou. Cílem této práce je rozšířit knihovnu libVATA o podporu této reprezentace včetně algoritmů pro některé operace nad touto reprezentací. Předložený text se v úvodu zabývá předpoklady nezbytnými pro pochopení konečných automatů a binárních rozhodovacích diagramů, které se využívají pro jejich symbolickou reprezentaci. Dále jsou uvedeny některé existující knihovny pro práci s konečnými automaty. Následuje jádro této práce, návrh symbolické reprezentace a operací nad ní, které jsou poté implementovány jako rozšíření zmíněné knihovny. Výsledky testů dokazují, že symbolická reprezentace je zajímavou alternativou explicitní reprezentace.

Abstract

In formal analysis we often encounter finite automata with a large amount of states over large alphabets. Their explicit representation can result in a state explosion and this problem can be solved by the use of symbolic representation that can manipulate a whole set of states at once. The aim of this work is to extend the libVATA library to support such a representation including algorithms for some operations on this representation. The presented text first deals with prerequisites necessary to undestand finite automata and binary decision diagrams used for their symbolic representation. Then it lists some existing libraries for work with finite automata. Next follows the core of this work, the design of a symbolic representation and operations on it, which are later implemented in the previously mentioned library. The test results proves that the symbolic representation is an interesting alternative to the explicit representation.

Klíčová slova

binární rozhodovací diagram, konečný automat, symbolická reprezentace, jazykové sjednocení, jazykový průnik, relace simulace, knihovna libVATA

Keywords

binary decision diagram, finite automaton, symbolic representation, language union, language intersection, simulation relation, libVATA library

Citace

Jiří Chromečka: Symbolická reprezentace konečných automatů, bakalářská práce, Brno, FIT VUT v Brně, 2014

Symbolická reprezentace konečných automatů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Ondřeje Lengála. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jiří Chromečka
18. května 2014

Poděkování

Děkuji především svému vedoucímu panu Ing. Ondřeji Lengálovi za jeho vstřícnost, odborné vedení a cenné rady u společných konzultací a také své rodině za podporu, kterou mi poskytli.

Obsah

1 Úvod **2**

2 Teoretický úvod **3**
2.1 Výroková logika . . . 3
2.2 Binární rozhodovací diagram . . . 4
2.3 Abeceda a jazyky . . . 9
2.4 Konečný automat . . . 10
2.5 Myhill-Nerodova věta . . . 11

3 Knihovny **12**
3.1 FSA . . . 12
3.2 MONA . . . 12
3.3 Timbuk . . . 12
3.4 LASH . . . 13
3.5 libVATA . . . 13
3.6 SA . . . 13

4 Návrh **14**
4.1 Struktura symbolické reprezentace . . . 14
4.2 Operace nad symbolickou reprezentací . . . 16

5 Implementace **21**
5.1 Symbolický konečný automat . . . 21
5.2 Operace nad konečným automatem . . . 22

6 Vyhodnocení **24**
6.1 Načtení automatu . . . 24
6.2 Jazykové sjednocení . . . 25
6.3 Jazykový průnik . . . 26
6.4 Relace simulace . . . 27
6.5 Diskuze výsledků . . . 28

7 Závěr **29**

A Obsah CD **31**
A.1 Struktura CD . . . 31

Kapitola 1

Úvod

V oblasti teoretické informatiky se jako prostředek pro řešení některých výpočetních problémů používají konečné automaty, které je důležité pro práci s nimi vhodným způsobem reprezentovat. V praxi se v některých případech vyskytují konečné automaty s velkým počtem stavů (např. automaty modelující systémy pro formální verifikaci) nebo s velkými abecedami (např. ukazatele do paměti, kde symbol představuje nejčastěji 32/64-bitové číslo), případně kombinace obojího. Explicitní reprezentace těchto automatů bývá problematická, protože může dojít ke stavové explozi, tedy exponenciálnímu růstu množství stavů v důsledku velkého množství kombinací. Jedním ze způsobů, jak se této explozi vyhnout, nebo ji alespoň omezit, je použití symbolické reprezentace, která přechodovou relaci automatu vyjadřuje pomocí jiných prostředků než explicitně, např. pomocí logických formulí.

Některé existující knihovny určené pro práci s konečnými automaty podporují tzv. semi-symbolickou reprezentaci, jež používá symbolickou reprezentaci symbolů abecedy, ale stavy jsou reprezentovány explicitně. Tato reprezentace se používá např. v nástroji MONA nebo knihovně libVATA a její použití je vhodné např. při implementaci rozhodovacích procedur pro některé fragmenty logiky (např. MSO nebo WS2S). Pro některé aplikace, jako je třeba výpočet relace simulace na automatu, však tato reprezentace vhodná není.

Tato práce se zaměřuje na plně symbolickou reprezentaci nedeterministických konečných automatů pomocí binárních rozhodovacích diagramů (BDD) a operace nad touto reprezentací, které budou implementovány jako rozšíření knihovny libVATA. Jedním z cílů práce je návrh a implementace algoritmu pro efektivní výpočet relace maximální simulace nad plně symbolickou reprezentací konečného automatu. Dále se uvažují operace jazykového sjednocení a jazykového průniku dvou konečných automatů.

V Kapitole 2 jsou uvedeny základní pojmy související s BDD a konečnými automaty. V Kapitole 3 jsou srovnány již existující knihovny pro práci s konečnými automaty. Kapitola 4 se zabývá návrhem symbolické reprezentace konečných automatů a algoritmů některých operací, které tato reprezentace podporuje. Kapitola 5 popisuje implementaci této reprezentace a algoritmů v prostředí knihovny libVATA. V Kapitole 6 následuje experimentální otestování a vyhodnocení dané implementace. Kapitola 7 shrnuje dosažené výsledky a diskutuje další možná rozšíření.

Kapitola 2

Teoretický úvod

V této kapitole je uvedena základní terminologie nezbytná k pochopení práce. Na úvod zmíníme výrokovou logiku, která byla převzata ze studijní opory k předmětu Matematická logika [1]. Poté přejdeme k binárním rozhodovacím diagramům, které čerpají z přednášek předmětů Formální analýza a verifikace [2] a Automated Reasoning [3]. Dále definujeme abecedu, jazyky a samotné konečné automaty, u kterých vycházíme z bakalářské práce Martina Hrušky [4] a studijní opory k předmětu Teoretická informatika [5] na FIT VUT.

2.1 Výroková logika

Výroková logika zkoumá způsoby tvorby složených výroků z daných jednoduchých výroků a závislost pravdivosti (resp. nepravdivosti) složeného výroku na pravdivosti výroků, z nichž je složen.

Buď P neprázdná množina symbolů, které nazýváme *výrokové proměnné*. Tyto symboly, které zpravidla značíme malými písmeny $p, q, \ldots$, mají úlohu jednoduchých výroků. Složené výroky vytváříme z jednoduchých výroků pomocí následujících *logických spojek*: negace $\neg$, konjunkce $\wedge$, disjunkce $\vee$, implikace $\implies$, ekvivalence $\iff$.

Symboly jazyka L_P výrokové logiky (nad množinou P) jsou prvky množiny P, logické spojky a závorky $(,)$. Úlohu složených výroků mají *výrokové formule* jazyka L_P definované následovně:

i Každá prvotní formule $p \in P$ je výroková formule.

ii Jsou-li A a B výrokové formule, pak $(\neg A), (A \wedge B), (A \vee B), (A \implies B), (A \iff B)$ jsou také výrokové formule.

iii Každá výroková formule vznikne konečným počtem užití pravidel i a ii.

Každá výroková formule je konečná posloupnost symbolů jazyka L_P, která vznikne podle předchozích pravidel.

Pravdivostní ohodnocení výrokových proměnných je libovolné zobrazení $v\colon P \to \{0, 1\}$, které každé výrokové proměnné $p \in P$ přiřadí hodnotu 0 (nepravda) nebo hodnotu 1 (pravda).

Logická funkce $f(x_1, \ldots, x_n)$ nad výrokovými proměnnými $x_1, \ldots, x_n$ je formule, v níž se vyskytují výrokové proměnné z množiny $\{x_1, \ldots, x_n\}$.

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \implies B$	$A \iff B$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

Tabulka 2.1: Sémantika logických spojek výrokové logiky

2.1.1 Shannonova expanze

Shannonova expanze je identita, která umožňuje vyjádřit logickou funkci nad n výrokovými proměnnými pomocí dvou logických funkcí nad $n-1$ výrokovými proměnnými. Postupnou aplikací této expanze na logickou funkci lze vytvořit binární rozhodovací diagram. Matematický zápis Shannonovy expanze je následující:

$$f(x_1, x_2, \ldots, x_n) \quad = \quad \big(\neg x_1 \wedge f(0, x_2, \ldots, x_n)\big) \vee \big(x_1 \wedge f(1, x_2, \ldots, x_n)\big). \tag{2.1}$$

2.2 Binární rozhodovací diagram

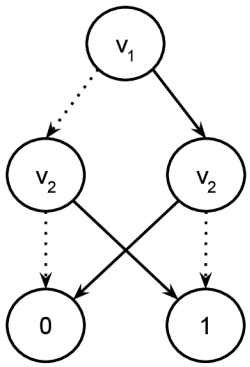
Binární rozhodovací diagram (dále jen BDD, Binary Decision Diagram) je grafová struktura, pomocí které lze efektivně reprezentovat logickou funkci. Vychází z binárních rozhodovacích stromů, které ale uchovávají redundantní informace. Z tohoto důvodu byly zavedeny uspořádané a redukované BDD, které odstraňují redundanci.

BDD je zakořeněný orientovaný souvislý acyklický graf reprezentující logickou funkci nad množinou výrokových proměnných *Var*. Každý neterminální uzel je označen proměnnou z množiny *Var* a má právě dva následníky dostupné přes hrany, které odpovídají hodnotám z množiny $\{0, 1\}$ zmíněné proměnné. Terminální uzly mají hodnotu z množiny $\{0, 1\}$, která představuje pravdivost logické funkce dané cestou v grafu od kořene přes hrany odpovídající proměnným z *Var* k příslušnému terminálnímu uzlu.

Formálně se jedná o sedmici $G = (N, T, var, low, high, root, val)$ nad množinou výrokových proměnných *Var*, kde:

- N je konečná množina neterminálních uzlů,
- T je konečná množina terminálních uzlů, přičemž $N \cap T = \varnothing$,
- *var* je zobrazení $N \to Var$, které každý neterminální uzel $n \in N$ označí proměnnou z množiny *Var*,
- *low* je zobrazení $N \to N \cup T$ určující následníka neterminálního uzlu $n \in N$ pro ohodnocení $var(n) = 0$,
- *high* je zobrazení $N \to N \cup T$ určující následníka neterminálního uzlu $n \in N$ pro ohodnocení $var(n) = 1$,
- *root* určuje kořenový uzel BDD, $root \in N \cup T$,
- *val* je zobrazení $T \to \{0, 1\}$, které každému terminálnímu uzlu $t \in T$ přiřadí hodnotu z množiny $\{0, 1\}$.

Hrany odpovídající funkci *low* budeme značit tečkovanou čarou a hrany odpovídající funkci *high* budeme značit plnou čarou. Obrázek 2.1 ukazuje, jak lze reprezentovat funkci $f(v_1, v_2) = \neg(v_1 \iff v_2)$ pomocí BDD.



Obrázek 2.1: BDD pro funkci $f(v_1, v_2) = \neg(v_1 \iff v_2)$

2.2.1 Uspořádaný binární rozhodovací diagram

Jsou-li uzly BDD uspořádané podle proměnných, mluvíme o uspořádaném BDD (dále jen OBDD, Ordered BDD). Uspořádání proměnných má významný vliv na velikost OBDD a nalezení optimálního uspořádání, pro které má OBDD nejmenší velikost, je klasifikováno jako NP-těžký problém.

Nechť $G = (N, T, var, low, high, root, val)$ je vzestupně (od kořene) uspořádané OBDD. Pak platí, že neterminální uzel, který je následníkem jiného uzlu, je označen takovou proměnnou, která je v uspořádání větší než proměnná rodičovského uzlu. Formálně pro $\forall n \in N$:

$$low(n) \in N \implies var(n) \prec var\big(low(n)\big), \tag{2.2}$$

$$high(n) \in N \implies var(n) \prec var\big(high(n)\big). \tag{2.3}$$

2.2.2 Redukovaný binární rozhodovací diagram

Pokud BDD neobsahuje žádné duplicitní a redundantní uzly, nazýváme ho redukované BDD (dále jen RBDD = Reduced BDD).

Nechť $G = (N, T, var, low, high, root, val)$ je RBDD. Pak platí, že (2.4) žádné dva různé neterminální uzly, které jsou označené stejnou proměnnou, nemají zároveň stejné levé i pravé následníky, dále (2.5) žádný neterminální uzel nemá shodného levého a pravého následníka a (2.6) žádné dva různé terminální uzly nemají stejnou hodnotu. Formálně pro $\forall n, m \in N$ a pro $\forall t, u \in T$:

$$n \neq m \wedge var(n) = var(m) \implies low(n) \neq low(m) \vee high(n) \neq high(m), \tag{2.4}$$

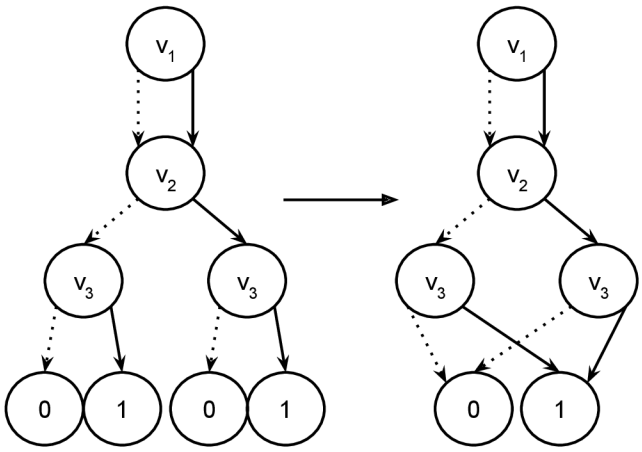
$$low(n) \neq high(n), \tag{2.5}$$

$$t \neq u \implies val(t) \neq val(u). \tag{2.6}$$

2.2.3 Operace *reduce*

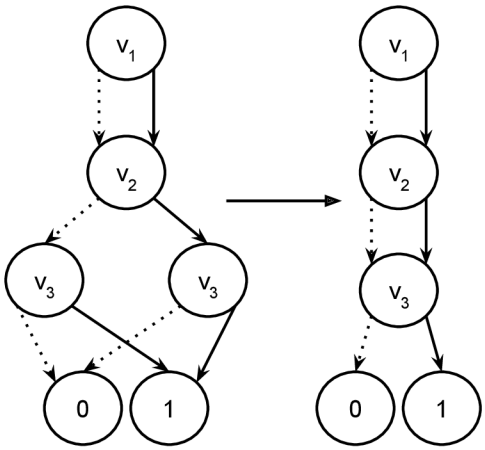
Operace *reduce* slouží k redukci BDD. Nechť f je logická funkce a B_f je neredukované BDD pro funkci f. Potom operací $B'_f := reduce(B_f)$ rozumíme redukci BDD B_f, jejíž výsledkem je RBDD B'_f. Postup této operace pro funkci $f(v_1, v_2, v_3) = (\neg v_1 \wedge \neg v_2 \wedge v_3) \vee (\neg v_1 \wedge v_2 \wedge v_3) \vee (v_1 \wedge \neg v_2 \wedge v_3) \vee (v_1 \wedge v_2 \wedge v_3)$ bude doplněn o grafickou ilustraci.

1. Nejdříve odstraníme všechny terminální uzly se stejnou hodnotou tak, že je sloučíme do jediného uzlu a všechny hrany, které vedly do těchto uzlů, přesměrujeme do nového uzlu (viz Obrázek 2.2).



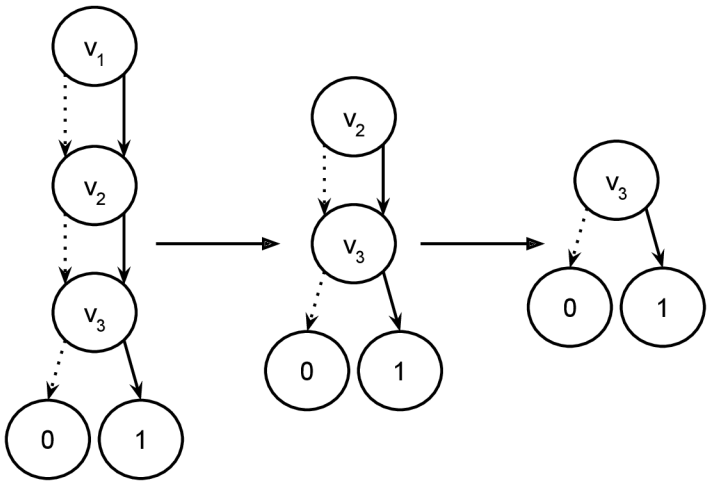
Obrázek 2.2: Odstranění duplicitních terminálních uzlů

2. Poté podobným způsobem odstraníme všechny neterminální uzly ohodnocené stejnou proměnnou se shodnými levými i pravými následníky tak, že je sloučíme do jediného uzlu a všechny hrany, které vedly do těchto uzlů, přesměrujeme do nového uzlu (viz Obrázek 2.3).



Obrázek 2.3: Odstranění duplicitních neterminálních uzlů

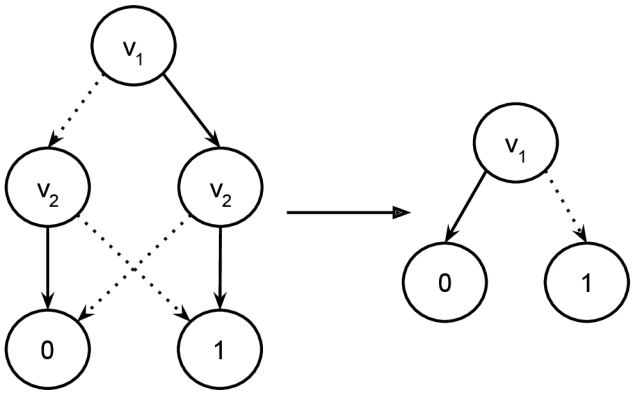
3. Nakonec odstraníme každý neterminální uzel se shodným levým i pravým následníkem tak, že všechny hrany, které vedly do tohoto uzlu, přesměrujeme do jeho následníka (viz Obrázek 2.4).



Obrázek 2.4: Odstranění redundantních neterminálních uzlů

2.2.4 Operace *restrict*

Pomocí operace *restrict* získáme nové BDD, kde je některé proměnné přiřazena konkrétní hodnota. Nechť c je konstanta, která může nabývat hodnot z množiny $\{0, 1\}$, f je logická funkce s proměnnými v_1 a v_2 a B_f je BDD pro funkcí f. Potom operací $B'_f := restrict(B_f, v_2, c)$ rozumíme restrikci BDD B_f dosazením konstanty c za proměnnou v_2, jejímž výsledkem je nové BDD B'_f. Na Obrázku 2.5 je ukázka této operace pro funkci $f(v_1, v_2) = v_1 \iff v_2$.



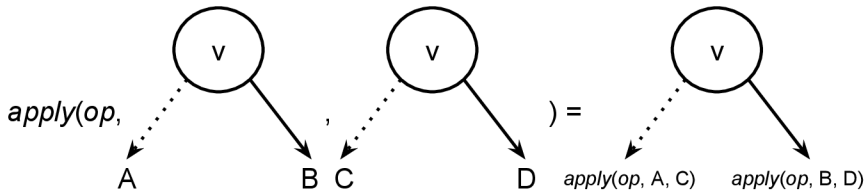
Obrázek 2.5: Dosazení konstanty 0 za proměnnou v_2

2.2.5 Operace *apply*

Operace *apply* vychází z Shannonovy expanze (viz Sekce 2.1.1).

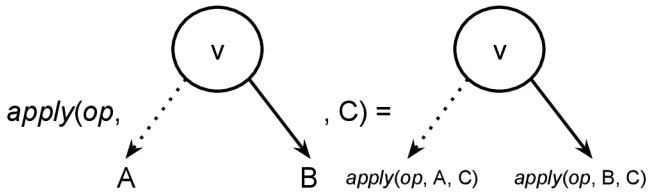
Nechť f a g jsou logické funkce, B_f a B_g jsou BDD pro funkce f a g v tomto pořadí a op je binární operace. Operací $B_{fg} := apply(op, B_f, B_g)$ rozumíme aplikaci operace op na BDD B_f a B_g, jejímž výsledkem je BDD B_{fg} takové, že B_{fg} reprezentuje funkci $fg = f\ op\ g$. Pro lepší pochopení rekurzivního průchodu nad dvěma BDD uvedeme čtyři graficky ilustrovaná pravidla.

1. Na Obrázku 2.6 je operace *apply* nad dvěma neterminálními uzly označenými toutéž proměnnou v, z nichž první má podstromy A a B a druhý má podstromy C a D. Výsledkem operace je neterminální uzel, jehož levý podstrom získáme rekurzivně operací *apply* nad podstromy A a C a pravý podstrom získáme rekurzivně operací *apply* nad podstromy B a D.



Obrázek 2.6: Krok *apply* pro dva uzly s toutéž proměnnou

2. Na Obrázku 2.7 je operace *apply* nad neterminálním uzlem s podstromy A a B označeném proměnnou v a uzlem C, který je 1) terminální, nebo 2) neterminální, přičemž platí, že $var(C) \succ v$. Výsledkem operace je neterminální uzel, jehož levý podstrom získáme rekurzivně operací *apply* nad podstromem A a uzlem C a pravý podstrom získáme rekurzivně operací *apply* nad podstromem B a uzlem C.

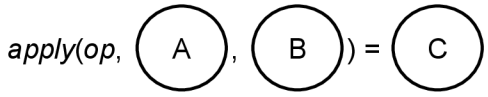


Obrázek 2.7: Krok *apply* pro podstrom a uzel

3. Na Obrázku 2.8 je, analogicky k předchozímu kroku, operace *apply* nad uzlem A, který je 1) terminální, nebo 2) neterminální, přičemž platí, že $var(A) \succ v$, a neterminálním uzlem s podstromy B a C označeném proměnnou v. Výsledkem operace je neterminální uzel, jehož levý podstrom získáme rekurzivně operací *apply* nad uzlem A a podstromem C a pravý podstrom získáme rekurzivně operací *apply* nad uzlem A a podstromem C.

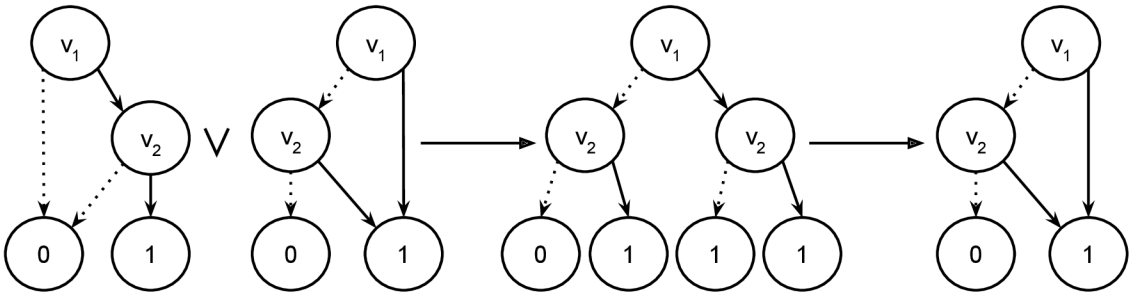
Obrázek 2.8: Krok *apply* pro uzel a podstrom

4. Na Obrázku 2.9 je operace *apply* nad dvěma terminálními uzly A a B. Výsledkem operace je terminální uzel C, přičemž $val(C) = val(A)\ op\ val(B)$.



Obrázek 2.9: Krok *apply* pro dva terminální uzly

Následuje ukázka $apply(\vee, f, g)$ pro $f(v_1, v_2) = v_1 \wedge v_2$ a $g(v_1, v_2) = v_1 \vee (\neg v_1 \wedge v_2)$ na Obrázku 2.10.



Obrázek 2.10: Ukázka $apply(\vee, f, g)$

2.3 Abeceda a jazyky

Abeceda Σ je konečná neprázdná množina symbolů, které značíme malými písmeny $a, b, \ldots$. Řetězec w nad abecedou Σ o délce $|w| = n$ je konečná posloupnost symbolů $w = a_1 \ldots a_n$, kde pro $\forall i\colon 1 \leq i \leq n$ platí, že $a_i \in \Sigma$. Speciálním případem řetězce je prázdný řetězec $\varepsilon \notin \Sigma$ s délkou $|\varepsilon| = 0$. Pro konkatenaci $\cdot$ nad řetězci ε, w a $x = b_1 \ldots b_m$ platí, že $\varepsilon \cdot w = w \cdot \varepsilon = w$ a $w \cdot x = a_1 \ldots a_n b_1 \ldots b_m$. Pro i-tou mocninu x^i nad řetězcem x platí, že $x^0 = \varepsilon$ a $x^{i+1} = x \cdot x^i$ pro $i \geq 1$. Množinu všech řetězců nad abecedou Σ vyjma prázdného řetězce ε označujeme Σ^+, přičemž $\Sigma^+ \cup \varepsilon = \Sigma^*$.

Jazyk L nad abecedou Σ je libovolná podmnožina Σ^*, tj. $L \subseteq \Sigma^*$. Potenční množinu všech jazyků nad abecedou Σ značíme 2^{Σ^*}. Pro konkatenaci $\cdot$ nad jazyky L_1 a L_2 nad abecedami Σ_1 a Σ_2 v tomto pořadí platí, že $L_1 \cdot L_2 = \{x \cdot y \mid x \in L_1, y \in L_2\}$. Pro i-tou mocninu L^i nad jazykem L platí následující:

- $L^0 = \{\varepsilon\}$,
- $L^{i+1} = L \cdot L^i$ pro $i \geq 1$,
- $L^* = \bigcup_{i \geq 0} L^i$,
- $L^+ = \bigcup_{i \geq 1} L^i$.

2.4 Konečný automat

Konečný automat (FA = Finite Automaton) je pětice $M = (Q, \Sigma, \Delta, I, F)$, kde:

- Q je konečná množina stavů,
- Σ je abeceda,
- $\Delta \subseteq Q \times \Sigma \times Q$ je přechodová relace,
- $I \subseteq Q$ je množina počátečních stavů a
- $F \subseteq Q$ je množina koncových stavů.

2.4.1 Přechody konečného automatu

Přechod konečného automatu $M = (Q, \Sigma, \Delta, I, F)$ mezi dvěma stavy $q_1, q_2 \in Q$ pomocí symbolu $a \in \Sigma$, přičemž $(q_1, a, q_2) \in \Delta$, značíme $q_1 \xrightarrow{a} q_2$. Běh řetězce $w = a_1 \dots a_{n-1} \in \Sigma^*$ ze stavu q_1 nad konečným automatem M je posloupnost přechodů konečného automatu M:

$$q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} q_n, \tag{2.7}$$

pro které platí:

- $q_1, \dots, q_n \in Q$,
- $q_m \xrightarrow{a_m} q_{m+1} \in \Delta$ pro $\forall m \colon 1 \leq m \leq n-1$.

Řetězec w je přijímán konečným automatem M, pokud $q_n \in F$. Jazyk $L(q) \subseteq \Sigma^*$ stavu $q \in Q$ je množina všech řetězců přijímaných automatem M ze stavu q. Jazyk $L(M) \subseteq \Sigma^*$ automatu M je množina všech řetězců přijímaných automatem M ze stavů $q \in I$.

2.4.2 Deterministický konečný automat

Konečný automat $M = (Q, \Sigma, \Delta, I, F)$ je deterministický, jestliže přechodová relace Δ neumožňuje pomocí jednoho symbolu přejít do více stavů zároveň:

$$\forall q \in Q, \forall a \in \Sigma \colon |\Delta(q, a)| \leq 1. \tag{2.8}$$

2.4.3 Jazykové sjednocení

Jazykovou operaci sjednocení dvou konečných automatů $M = (Q_M, \Sigma, \Delta_M, I_M, F_M)$ a $N = (Q_N, \Sigma, \Delta_N, I_N, F_N)$ se společnou abecedou Σ definujeme následovně:

$$M \cup N = (Q_M \cup Q_N, \Sigma, \Delta_M \cup \Delta_N, I_M \cup I_N, F_M \cup F_N). \tag{2.9}$$

Platí, že jazyk $L(M \cup N) = L(M) \cup L(N)$.

2.4.4 Jazykový průnik

Jazykovou operaci průniku dvou konečných automatů $M = (Q_M, \Sigma, \Delta_M, I_M, F_M)$ a $N = (Q_N, \Sigma, \Delta_N, I_N, F_N)$ se společnou abecedou Σ definujeme následovně:

$$M \cap N = (Q_M \times Q_N, \Sigma, \Delta, I_M \times I_N, F_M \times F_N), \tag{2.10}$$

kde $\Delta = \{(q_1, r_1) \xrightarrow{a} (q_2, r_2) \mid q_1 \xrightarrow{a} q_2 \in \Delta_M \wedge r_1 \xrightarrow{a} r_2 \in \Delta_N\}$.

Platí, že jazyk $L(M \cap N) = L(M) \cap L(N)$.

2.4.5 Relace simulace

Relace simulace $\leq$ nad konečným automatem $M = (Q, \Sigma, \Delta, I, F)$ je binární relace nad stavy z množiny Q, tedy $\leq \subseteq Q \times Q$. $q \leq r$ znamená, že pokud je stav q simulován stavem r, potom pro každý přechod ze stavu q pomocí symbolu $a \in \Sigma$ do stavu q' existuje přechod ze stavu r pomocí stejného symbolu a do stavu r' pro $q', r' \in Q$, který zachovává relaci simulace $q' \leq r'$ a dále platí, že pokud $q \in F$, potom i $r \in F$:

$$q \leq r \implies \forall a \forall q' \,.\, \Big(q \xrightarrow{a} q' \implies \exists r' \,.\, \big(r \xrightarrow{a} r' \wedge q' \leq r'\big)\Big) \wedge \Big(q \in F \implies r \in F\Big). \tag{2.11}$$

2.5 Myhill-Nerodova věta

Pomocí Myhill-Nerodovy věty se dá dokázat existence jedinečného (až na izomorfismus) minimálního deterministického konečného automatu.

Relace ekvivalence $\sim$ je pravou kongruencí, pokud je zprava invariantní pro libovolný řetězec nad abecedou Σ: $\forall u, v, w \in \Sigma^* \,.\, u \sim v \implies uw \sim vw$.

Má-li rozklad $\Sigma/\sim$ konečně mnoho tříd, je kongruence konečného indexu. Věta tvrdí, že pro jazyk L nad abecedou Σ a relaci $\sim_L$ zvanou prefixová ekvivalence pro jazyk L jsou tvrzení

- jazyk L je přijímaný deterministickým konečným automatem,
- jazyk L je sjednocením některých tříd rozkladu určeného pravou kongruencí na Σ^* s konečným indexem a
- relace $\sim_L$ má konečný index,

ekvivalentní.

Počet stavů minimálního deterministického konečného automatu přijímajícího jazyk L je roven indexu relace $\sim_L$.

Kapitola 3

Knihovny

Pro nasazení konečných automatů je musíme vhodným způsobem reprezentovat a implementovat nad nimi potřebné operace. Tato kapitola pojednává o knihovnách, které se konečnými automaty nějakým způsobem zabývají. Knihovny se využívají ve formální verifikaci, analýze a dalších odvětvích (nejen) teoretické informatiky.

3.1 FSA

Knihovna Finite State Automata Utilities [6] je napsaná v jazyce Prolog a distribuována pod licencí GNU GPL. Jedná se o sbírku nástrojů pro práci s regulárními výrazy, konečnými automaty a převodníky. Zahrnuje převod regulárních výrazů na konečné automaty, determinizaci, minimalizaci, kompozici, doplněk, průnik, Kleeneho uzávěr a další operace. Jsou k dispozici různé vizualizační nástroje pro zobrazení automatů a také interprety automatů pro jejich použití. Konečné automaty mohou být také kompilovány do samostatných programů v jazyce C.

3.2 MONA

MONA [7], od slova monadický, je implementace rozhodovací procedury pro slabou druhořádovou monadickou teorii s 1/2 následníky (WS1S/WS2S). Formule v této teorii mohou vyjadřovat vyhledávací vzory, temporální vlastnosti reaktivních systémů, omezení derivačních stromů a další.

Nástroj převede formuli φ na konečný stromový automat A_φ a testuje platnost či splnitelnost φ tím, že analyzuje automat A_φ. Nástroj používá semi-symbolickou reprezentaci (viz Kapitola 1).

3.3 Timbuk

Timbuk [8] je sbírka nástrojů v jazyce OCaml, která se využívá pro analýzu dosažitelnosti u přepisovacích systémů s termy.

Její starší verze (2.2) poskytuje funkce pro práci se stromovými automaty, mezi jinými například operace sjednocení, průniku, testování inkluze, determinizace a další.

3.4 LASH

LASH [9] je soubor nástrojů určených pro reprezentaci nekonečných množin a procházení nekonečných stavových prostorů. Je založen na konečných automatech pro reprezentaci a manipulaci s nekonečnými množinami hodnot různých datových typů.

LASH obsahuje knihovny v jazyce C poskytující prostředky pro konstrukci a manipulaci s konečnými automaty nad konečnými a nekonečnými řetězci, pro reprezentaci lineárních rovnic nad celočíselnými i reálnými čísly a další.

3.5 libVATA

Knihovna libVATA [10] je vysoce optimalizovaná pro práci s nedeterministickými konečnými a stromovými automaty. V současnosti podporuje explicitní a semi-symbolickou reprezentaci automatů a primárním cílem této práce je rozšířit ji o podporu plně symbolické reprezentace. Knihovna poskytuje vlastní běžné i více-terminálové BDD, kterých využívá zmíněná semi-symbolická reprezentace.

3.6 SA

Knihovna SA [11] v jazyce OCaml slouží k práci se stromovými automaty a přechodovými systémy s návěštími (LTS = Labelled Transition Systems).

Podporuje výpočet dopředné a zpětné simulace nad stromovými automaty a LTS.

Kapitola 4

Návrh

V případě explicitní reprezentace konečných automatů nad velkou abecedou, anebo s velkým množstvím stavů, mohou exponenciálně růst paměťové a časové nároky výpočetních operací a dochází ke stavové explozi. Příčinou je manipulace s každým vzniklým stavem samostatně. Z tohoto důvodu zavadíme plně symbolickou reprezentaci, která umožňuje pracovat se skupinami stavů a symbolů, snižuje tak paměťové nároky a dovoluje implementaci efektivních operací nad touto reprezentací.

V této kapitole uvedeme, jak efektivně implementovat symbolickou reprezentaci konečných automatů pomocí BDD, a popíšeme návrh algoritmů pro operace jazykového sjednocení, jazykového průniku a výpočtu relace simulace na této reprezentaci.

4.1 Struktura symbolické reprezentace

Základní myšlenkou návrhu reprezentace je, že BDD budou reprezentovat množinu počátečních stavů I, množinu koncových stavů F a přechodovou relaci Δ konečného automatu. Každý stav a symbol konečného automatu zobrazíme na posloupnost hodnot z množiny $\{0, 1\}$, která představuje ohodnocené výrokové proměnné z množiny X (pro stavy) a Y (pro symboly). Pro zobrazení využijeme následující funkce:

$$enc_Q \colon Q \to 2^{X \to \{0,1\}}, \tag{4.1}$$

$$enc_\Sigma \colon \Sigma \to 2^{Y \to \{0,1\}}. \tag{4.2}$$

Každý stav, symbol nebo relace potom v BDD představuje cestu od kořene přes hrany odpovídající proměnným z množin X, Y až k listovým uzlům v závislosti na uspořádání proměnných.

4.1.1 Přechodová relace

BDD, které představuje přechodovou relaci Δ konečného automatu, se analogicky k Δ dělí na množinu stavů Q_1, abecedu Σ a množinu stavů Q_2. Každý přechod je v BDD pro logickou funkci f reprezentován následovně:

$$q \xrightarrow{a} r \in \Delta \iff f(enc_Q(q) \cup enc_\Sigma(a) \cup enc_Q(r)') = 1. \tag{4.3}$$

4.1.2 Množina počátečních stavů

BDD, které představuje množinu počátečních stavů I konečného automatu, je tvořené jedinou množinou stavů Q. Každý stav je v BDD pro logickou funkci f reprezentován následovně:

$$q_I \in I \iff f(enc_Q(q_I)) = 1. \tag{4.4}$$

4.1.3 Množina koncových stavů

BDD, které představuje množinu koncových stavů F konečného automatu, sestává také z jediné množiny stavů Q. Každý stav je v BDD pro logickou funkci f reprezentován následovně:

$$q_F \in F \iff f(enc_Q(q_F)) = 1. \tag{4.5}$$

4.1.4 Příklad symbolicky reprezentovaného konečného automatu

Pro lepší pochopení uvedeme příklad celého procesu symbolické reprezentace konečného automatu. Nechť $A = (Q, \Sigma, \Delta, I, F)$ je konečný automat, kde:

- $Q = \{q, r\}$,
- $\Sigma = \{a, b\}$,
- $\Delta = \{q \xrightarrow{a} q, q \xrightarrow{b} r, r \xrightarrow{a} r, r \xrightarrow{b} r\}$,
- $I = \{q\}$,
- $F = \{r\}$.

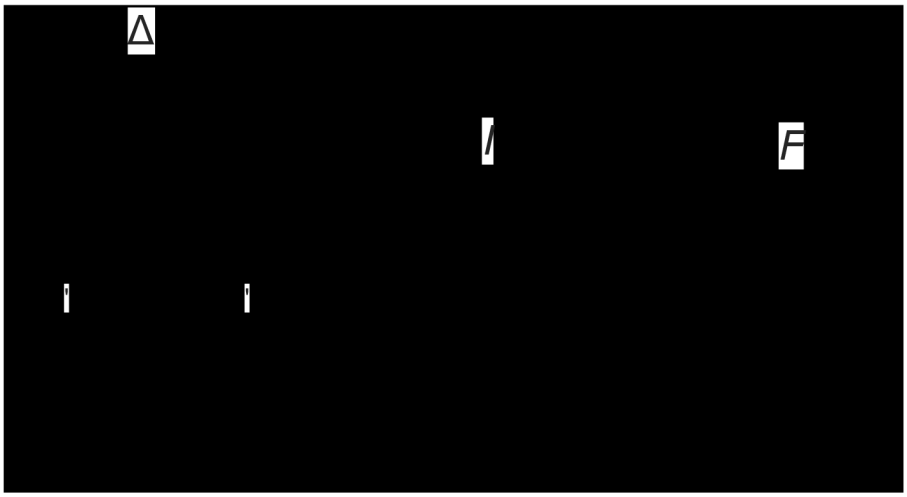
Nechť $X = \{x\}$, $Y = \{y\}$ a $X' = \{x'\}$ jsou množiny výrokových proměnných. Pomocí kódovacích funkcí 4.1 a 4.2 převedeme každý stav a symbol konečného automatu A na jejich symbolický tvar následovně:

- $enc_Q(q) = \{x \mapsto 0\}$, $enc_Q(r) = \{x \mapsto 1\}$,
- $enc_Q(q)' = \{x' \mapsto 0\}$, $enc_Q(r)' = \{x' \mapsto 1\}$,
- $enc_\Sigma(a) = \{y \mapsto 0\}$, $enc_\Sigma(b) = \{y \mapsto 1\}$.

Nechť B_f pro funkci f reprezentuje přechodovou relaci Δ, B_g pro funkci g reprezentuje množinu počátečních stavů I a B_h pro funkci h reprezentuje množinu koncových stavů F konečného automatu A. Potom funkční hodnoty funkcí f, g a h podle vztahů 4.3, 4.4 a 4.5 budou 0 kromě následujících případů:

- $f(enc_Q(q) \cup enc_\Sigma(a) \cup enc_Q(q)') = 1$,
- $f(enc_Q(q) \cup enc_\Sigma(b) \cup enc_Q(r)') = 1$,
- $f(enc_Q(r) \cup enc_\Sigma(a) \cup enc_Q(r)') = 1$,
- $f(enc_Q(r) \cup enc_\Sigma(b) \cup enc_Q(r)') = 1$,
- $g(enc_Q(q)) = 1$,
- $h(enc_Q(r)) = 1$.

Na Obrázku 4.1 je ukázka struktur výše definovaných BDD B_f, B_g a B_h.



Obrázek 4.1: Konečný automat A reprezentovaný pomocí BDD B_f, B_g a B_h

4.2 Operace nad symbolickou reprezentací

Operace nad konečnými automaty lze efektivně realizovat pomocí operací nad BDD. Níže uvedeme algoritmy pro operace jazykového sjednocení, jazykového průniku a výpočtu relace simulace.

Pro potřeby algoritmů zavedeme funkci $var(Q)$, která vrací velikost množiny výrokových proměnných X určené pro reprezentaci prvků množiny Q (viz rovnice 4.1 a 4.2).

Dále definujeme funkci pro vytvoření BDD $createBDD(v, l, h)$, která vytvoří neterminální uzel n, pro který platí následující: $var(n) = v$, $low(l) = v$ a $high(n) = h$.

4.2.1 Jazykové sjednocení

Jazykové sjednocení (viz Sekce 2.4.3) dvou konečných automatů $A = (Q_A, \Sigma, \Delta_A, I_A, F_A)$ a $B = (Q_B, \Sigma, \Delta_B, I_B, F_B)$ vyžaduje takové disjunktní přejmenování stavů obou automatů, aby $Q_A \cap Q_B = \varnothing$.

Nechť B_f je BDD pro funkci f. Pro přidání výrokové proměnné x s hodnotou y zavedeme funkci *AddVariable*(BDD B_f, Variable x, Value y), která vrací BDD $B_{f'}$ pro funkci f': $f \wedge x$, kde $f'(\{x \mapsto y\}) = 1$.

Funkce AddVariable(BDD B_f, Variable x, Value y)

```
1 begin
2     if  y = true then
3         BDD b := createBDD(x, false, true)
4     else
5         BDD b := createBDD(x, true, false)
6     end if
7     BDD B_f' := apply(∧, B_f, b)
8     return B_f'
9 end
```

Dále zavedeme funkci *RenameStates*(Automaton A, Automaton B), která pomocí funkce *AddVariable* rozšíří množinu výrokových proměnných X (viz vztah 4.1) o takový počet výrokových proměnných (ohodnocené '0' v případě prvního automatu a '1' v případě druhého), aby symbolická reprezentace stavů výsledného automatu pokryla všechny stavy obou vstupních automatů A a B. V případě BDD pro přechodovou relaci Δ voláme funkci *AddVariable* dvakrát, protože každý přechod sestává ze dvou stavů.

Funkce RenameStates(Automaton A, Automaton B)

begin
  $m := (var(Q_C) \geq var(Q_D))\ ?\ 1 : (var(Q_D) - var(Q_C) + 1)$
  $n := (var(Q_D) \geq var(Q_C))\ ?\ 1 : (var(Q_C) - var(Q_D) + 1)$
  for $i := var(Q_A)$ **to** m **do**
    BDD $b :=$ `AddVariable`$(\varnothing, x_i, 0)$
    $b' :=$ `AddVariable`$(b, x'_i, 0)$
    BDD $\Delta'_A := apply(\wedge, \Delta_A, b')$
    BDD $I'_A :=$ `AddVariable`$(I_A, x_i, 0)$
    BDD $F'_A :=$ `AddVariable`$(F_A, x_i, 0)$
  end for
  for $i := var(Q_B)$ **to** n **do**
    BDD $b :=$ `AddVariable`$(\varnothing, x_i, 1)$
    $b' :=$ `AddVariable`$(b, x'_i, 1)$
    BDD $\Delta'_B := apply(\wedge, \Delta_B, b')$
    BDD $I'_B :=$ `AddVariable`$(I_B, x_i, 1)$
    BDD $F'_B :=$ `AddVariable`$(F_B, x_i, 1)$
  end for
  Automaton $A' := (Q_A, \Sigma, \Delta'_A, I'_A, F'_A)$
  Automaton $B' := (Q_B, \Sigma, \Delta'_B, I'_B, F'_B)$
  return A', B'
end

Postup operace jazykového sjednocení ukazuje Algoritmus 1.

Algoritmus 1: Sjednocení dvou konečných automatů

Input: Automaton $A = (Q_A, \Sigma, \Delta_A, I_A, F_A)$, Automaton $B = (Q_B, \Sigma, \Delta_B, I_B, F_B)$
Output: Automaton $C = (Q_\cup, \Sigma, \Delta_\cup, I_\cup, F_\cup)$

begin
  Automaton $C, D :=$ `RenameStates`(A, B)
  Set $Q_\cup := Q_C \cup Q_D$
  BDD $\Delta_\cup := apply(\vee, \Delta_C, \Delta_D)$
  BDD $I_\cup := apply(\vee, I_C, I_D)$
  BDD $F_\cup := apply(\vee, F_C, F_D)$
  Automaton $C := (Q_\cup, \Sigma, \Delta_\cup, I_\cup, F_\cup)$
  return C
end

4.2.2 Jazykový průnik

Jazykový průnik (viz Sekce 2.4.4) dvou konečných automatů $A = (Q_A, \Sigma, \Delta_A, I_A, F_A)$ a $B = (Q_B, \Sigma, \Delta_B, I_B, F_B)$ zahrnuje kartézský součin $Q_A \times Q_B$. V BDD jej obdržíme přejmenováním množiny proměnných X (viz vztah 4.1) a pomocí *apply* s operací $\wedge$.

Definujeme tedy funkci *renameVars*(BDD b, Mapping f), která v BDD b přejmenuje množinu výrokových proměnných zobrazením $f\colon X \mapsto X_B$.

Postup operace jazykového průniku znázorňuje Algoritmus 2.

Algoritmus 2: Průnik dvou konečných automatů

Input: Automaton $A = (Q_A, \Sigma, \Delta_A, I_A, F_A)$, Automaton $B = (Q_B, \Sigma, \Delta_B, I_B, F_B)$
Output: Automaton $C = (Q_\cap, \Sigma, \Delta_\cap, I_\cap, F_\cap)$

begin
  BDD Δ'_B := *renameVars*$(\Delta_B, \{x \mapsto x_B, x' \mapsto x'_B\})$
  BDD I'_B := *renameVars*$(I_B, \{x \mapsto x_B\})$
  BDD F'_B := *renameVars*$(F_B, \{x \mapsto x_B\})$
  Set $Q_\cap := Q_A \times Q_B$
  BDD $\Delta_\cap$:= *apply*$(\wedge, \Delta_A, \Delta'_B)$
  BDD $I_\cap$:= *apply*$(\wedge, I_A, I'_B)$
  BDD $F_\cap$:= *apply*$(\wedge, F_A, F'_B)$
  Automaton $C := (Q_\cap, \Sigma, \Delta_\cap, I_\cap, F_\cap)$
  return C
end

4.2.3 Relace simulace

Výpočet relace simulace (viz Sekce 2.4.5) nad konečným automatem $A = (Q, \Sigma, \Delta, I, F)$ využívá pro přejmenování výrokových proměnných funkci *renameVars* (viz Sekce 4.2.2).

Nechť B_f je BDD pro funkci $f(x_0, \ldots, x_n)$. Nejdříve zavedeme funkci *Exists*(BDD B_f, Set X), která vrací BDD pro funkci $f'(x_0, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = f(x_0, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \vee f(x_0, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$. Poté funkci zobecníme na *Exists*(BDD B_f, Set X) pro postupnou aplikaci původní *Exists* na všechny výrokové proměnné z množiny X, tedy *Exists*(BDD B_f, Set $X = \{x_0, \ldots, x_n\}$) = *Exists*(*Exists*(\ldots(*Exists*$(B_f, x_0)) \ldots), x_n)$.

Funkce Exists(BDD B_f, Set X)

begin
  BDD $B'_f = B_f$
  foreach *Variable* $x \in X$ **do**
    BDD l := *restrict*$(B'_f, x, 0)$
    BDD r := *restrict*$(B'_f, x, 1)$
    BDD B'_f := *apply*$(\vee, l, r)$
  end foreach
  return B'_f
end

Dále zavedeme funkci *ForAll*(BDD B_f, Variable x_i), která analogicky k funkci *Exists* vrací BDD pro funkci $f'(x_0, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = f(x_0, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \wedge f(x_0, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$. Tuto funkci také zobecníme na *ForAll*(BDD B_f, Set X) pro postupnou aplikaci původní *ForAll* na všechny výrokové proměnné z množiny X, tedy $\mathit{ForAll}(\text{BDD } B_f, \text{ Set } X = \{x_0, \ldots, x_n\}) = \mathit{ForAll}(\mathit{ForAll}(\ldots(\mathit{ForAll}(B_f, x_0))\ldots), x_n)$.

Funkce ForAll(BDD B_f, Set X)

```
1  begin
2      BDD B'_f = B_f
3      foreach Variable x ∈ X do
4          BDD l := restrict(B'_f, x, 0)
5          BDD r := restrict(B'_f, x, 1)
6          BDD B'_f := apply(∧, l, r)
7      end foreach
8      return B'_f
9  end
```

Nyní zavedeme funkci *Init*(Automaton A), pomocí které získáme BDD pro počáteční relaci simulace $\leq_0$ nad konečným automatem A tak, aby platilo:

$$q_1 \leq_0 q_4 \implies \forall a \,.\, (\exists q_2 \,.\, q_1 \xrightarrow{a} q_2 \implies \exists q_3 \,.\, q_4 \xrightarrow{a} q_3) \wedge (q_1 \in F \implies q_4 \in F). \tag{4.6}$$

V následujícím algoritmu funkce *Init* použijeme pro reprezentaci množin stavů a symbolů $Q_1, Q_2, Q_3, Q_4, \Sigma$ množiny výrokových proměnných $X = \{x_0, \ldots, x_n\}, X' = \{x'_0, \ldots, x'_n\}, X'' = \{x''_0, \ldots, x''_n\}, X''' = \{x'''_0, \ldots, x'''_n\}, Y = \{y_0, \ldots, y_n\}$ v tomto pořadí.

Funkce Init(Automaton A)

```
1  begin
2      BDD lhs := Exists(Δ, X')                                  // ∃q2 . q1 -a-> q2
3      BDD rhs := renameVars(lhs, {x ↦ x'''})                    // ∃q3 . q4 -a-> q3
4      result := apply(⟹, lhs, rhs)                             // ∃q2 . q1 -a-> q2 ⟹ ∃q3 . q4 -a-> q3
5      result := ForAll(result, Y)                               // ∀a . (∃q2 . q1 -a-> q2 ⟹ ∃q3 . q4 -a-> q3)
6      lhs := F                                                  // q1 ∈ F
7      rhs := renameVars(lhs, {x ↦ x'''})                        // q4 ∈ F
8      BDD temp := apply(⟹, lhs, rhs)                           // q1 ∈ F ⟹ q4 ∈ F
9      BDD ≤0 := apply(∧, result, temp)                          // ∀a . (...) ∧ (q1 ∈ F ⟹ q4 ∈ F)
10     return ≤0
11 end
```

Pomocí funkce *Iterate*(Automaton A, BDD $\leq_i$) zjemňujeme relaci simulace $\leq_i$ nad konečným automatem A, dokud nedosáhneme pevného bodu. Předchozí vztah 4.6 upravíme následovně:

$$q_1 \leq_{i+1} q_4 \implies \forall a \forall q_2 \,.\, \Big(q_1 \xrightarrow{a} q_2 \implies \exists q_3 \,.\, (q_4 \xrightarrow{a} q_3 \wedge q_2 \leq_i q_3)\Big) \wedge \Big(q_1 \in F \implies q_4 \in F\Big). \tag{4.7}$$

Analogicky k funkci *Init* použijeme v následujícím algoritmu funkce *Iterate* pro reprezentaci množin stavů a symbolů $Q_1, Q_2, Q_3, Q_4, \Sigma$ množiny výrokových proměnných $X = \{x_0, \ldots, x_n\}, X' = \{x'_0, \ldots, x'_n\}, X'' = \{x''_0, \ldots, x''_n\}, X''' = \{x'''_0, \ldots, x'''_n\}, Y = \{y_0, \ldots, y_n\}$ v tomto pořadí.

Funkce Iterate(Automaton A, BDD $\leq_i$)

```
1  begin
2      BDD lhs := renameVars(Δ, {x ↦ x''', x' ↦ x''})          // q4 -a-> q3
3      BDD rhs := renameVars(≤_i, {x ↦ x', x' ↦ x''})          // q2 ≤_i q3
4      BDD result := apply(∧, lhs, rhs)                        // q4 -a-> q3 ∧ q2 ≤_i q3
5      rhs := Exists(result, X'')                              // ∃q3 . (q4 -a-> q3 ∧ q2 ≤_i q3)
6      lhs := Δ                                                // q1 -a-> q2
7      result := apply( ⟹ , lhs, rhs)                         // q1 -a-> q2 ⟹ ∃q3 . (q4 -a-> q3 ∧ q2 ≤_i q3)
8      result := ForAll(result, X')                            // ∀q2 . (q1 -a-> q2 ⟹ ∃q3 . (q4 -a-> q3 ∧ q2 ≤_i q3))
9      result := ForAll(result, Y)                             // ∀a∀q2 . (...)
10     lhs := F                                                // q1 ∈ F
11     rhs := renameVars(F, {x ↦ x'''})                        // q4 ∈ F
12     BDD temp := apply( ⟹ , lhs, rhs)                       // q1 ∈ F ⟹ q4 ∈ F
13     BDD ≤_{i+1} := apply(∧, result, temp)                   // ∀a∀q2 . (...) ∧ (q1 ∈ F ⟹ q4 ∈ F)
14     return ≤_{i+1}
15 end
```

Celý výpočet maximální relace simulace uvádí Algoritmus 3.

Algoritmus 3: Výpočet relace simulace

Input: Automat $A = (Q, \Sigma, \Delta, I, F)$
Output: BDD $\leq_i$

```
1  begin
2      i := 0
3      ≤_i := Init(A)
4      i := i + 1
5      ≤_i := Iterate(A, ≤_{i-1})
6      while ≤_i ≠ ≤_{i-1} do
7          i := i + 1
8          ≤_i := Iterate(A, ≤_{i-1})
9      end while
10     return ≤_i
11 end
```

Kapitola 5

Implementace

Tato kapitola popisuje implementaci plně symbolické reprezentace (viz Kapitola 4) v knihovně libVATA (viz Sekce 3.5). V následujícím popisu objektově-orientovaného přístupu k implementaci se nejdříve zaměříme na strukturu plně symbolického konečného automatu, a poté na jednotlivé operace jazykového sjednocení, jazykového průniku a výpočtu relace simulace.

5.1 Symbolický konečný automat

Hlavní třída `SymbolicFiniteAut` je zjednodušeným rozhraním pro práci se symbolickým konečným automatem. Obsahuje základní datové typy a funkce pro I/O operace. Také si přes objekt `core_` udržuje jádro automatu, které obsahuje všechny výpočetní operace. Všechny stavy a symboly automatu jsou z externí reprezentace (řetězec) převedeny na interní reprezentaci (číslo) pomocí již implementované třídy `TwoWayDict`, která představuje obousměrný slovník.

Šablonová třída `SymbolicLoadableAut` rozšiřuje třídu s jádrem automatu o vstupně-výstupní operace zahrnující jak explicitní, tak symbolické načtení a serializaci automatu (viz Sekce 5.2.1 a 5.2.2).

Pro reprezentaci BDD byla využita již implementovaná třída `OndriksMTBDD`. Rozhraním mezi ní a jádrem automatu je třída `SymbolicFiniteAutBDD`. Tato třída k symbolické reprezentaci prvků v BDD využívá strukturu `SymbolicVarAsgn`, binární pole rozšířené o hodnotu 'X', která představuje obě hodnoty '0' i '1'. Členy této třídy jsou `vars_`, uchovávající počet proměnných reprezentující v BDD jeden prvek, a `mtbdd_`, objekt zmíněné třídy `OndriksMTBDD`. Dále tato třída implementuje funktory s operací *apply*, které dědí již implementovanou třídu `Apply1Functor` a vlastní třídu `BoolApply2Functor`: unární funktor negace `NegApplyFunctor`, binární funktory disjunkce `OrApplyFunctor`, konjunkce `AndApplyFunctor`, implikace `ImplicApplyFunctor` a ekvivalence `EquivApplyFunctor`. Mezi hlavní metody třídy patří `AddAssignment`, která vloží do BDD další prvek, a `GetAllAssignments` pro získání všech prvků v BDD. Také jsou zde funkce pro převod ze symbolické do interní reprezentace (`FromSymbolic`), vytvoření symbolického přechodu z daných stavů a symbolů (`MergeTransition`) a jeho zpětné rozdělení na stavy a symboly (`SplitTransition`).

Jádro automatu představuje třída `SymbolicFiniteAutCore`. Mezi její členy patří `stateVars_` a `symbolVars_`, které uchovávají počet proměnných reprezentujících stavy a symboly, a dále tři objekty třídy `SymbolicFiniteAutBDD`: `transitions_`, `initialStates_` a `finalStates_` pro přechodovou relaci, množinu počátečních a koncových stavů v tomto pořadí.

5.2 Operace nad konečným automatem

V této sekci popíšeme implementaci importu a exportu konečného automatu, sjednocení a průnik dvou konečných automatů a výpočet relace simulace nad stavy konečného automatu. Relace simulace se dále může využít například pro redukci počtu stavů konečného automatu.

5.2.1 Načtení automatu

Načtení (neboli import) automatu dělíme na 1) přímé a nepřímé 2) explicitní a symbolické. Přímé načtení je na rozdíl od nepřímého pouze symbolické a využívá tří funkcí: `AddTransition` pro přídání přechodu, `AddInitialState` pro vkládání počátečního stavu a `AddFinalState` pro uložení koncového stavu. Nazýváme ho přímé, protože obchází parsování řetězce reprezentujícího automat a umožňuje vkládat prvky automatu přímo do BDD v požadovaném symbolickém tvaru.

Nepřímý import automatu využívá již implementované struktury `AutDescription`, do které parsuje řetězec automat popisující, v současnosti v jednotném vstupním formátu knihovny Timbuk. Dělíme jej na explicitní a symbolické načtení.

Explicitní načtení automatu vychází ze jmenné reprezentace všech stavů a symbolů, které nejdříve převedeme na číselnou, a poté na symbolickou reprezentaci. K tomuto slouží metoda `LoadFromAutDescExplicit`, ve které se prvním průchodem výše zmíněné struktury `AutDescription` načtou všechny stavy a symboly, zjistí se optimální počet proměnných pro jejich symbolickou reprezentaci a následně se druhým průchodem uloží do příslušných BDD.

U symbolického načtení automatu pomocí metody `LoadFromAutDescSymbolic` stačí jediný průchod, protože všechny stavy a symboly jsou v symbolickém tvaru a lze je jednoduše vložit do daných BDD.

5.2.2 Serializace automatu

Opět rozlišujeme explicitní a symbolickou serializaci (neboli export) automatu. V případě explicitní serializace je nutné dodat slovník `TwoWayDict` zmíněný výše, který převede číselnou reprezentaci stavů a symbolů na jmennou. Funkce `DumpToAutDescExplicit` zpracuje všechny přechody, počáteční a koncové stavy z daných BDD a uloží je do struktury `AutDescription`.

Symbolický export automatu všechny stavy a symboly po načtení z BDD přímo v podobě řetězce umístí do struktury `AutDescription`.

5.2.3 Jazykové sjednocení

Jazykové sjednocení automatů, které vychází z Algoritmu 1, implementuje metoda `Union`. Změnu indexace proměnných reprezentujících stavy automatu umožňuje funkce `Rename` ve třídě `OndriksMTBDD`. Samotné sjednocení lze jednoduše provést pomocí funktoru `OrApplyFunctor`.

Pro potřeby explicitní serializace sjednoceného automatu lze pomocí metody `GenUnionTransl` vygenerovat mapování reindexovaných stavů na původní stavy. A pomocí funkce `CreateUnionStringToStateMap`, která pracuje se zmíněným mapováním stavů, obdržíme nový slovník pro překlad stavů.

5.2.4 Jazykový průnik

Jazykový průnik dvou konečných automatů je postaven na Algoritmu 2 a realizuje jej metoda `Intersection`. Stejně jako u sjednocení využívá pro úpravu indexů proměnných reprezentujících stavy automatu funkci `Rename`. Pro operaci průniku je určen funktor `AndApplyFunctor`.

Podobně jako u sjednocení, i zde je implementována samostatná funkce `GenIsectTransl`, která vytvoří mapování stavů v kartézském součinu na dvojice původních stavů. Prostřednictvím funkce `CreateProductStringToStateMap` lze potom získat nový slovník pro překlad stavů.

5.2.5 Relace simulace

Výpočet relace simulace je založen na Algoritmu 3 a je implementován metodou `ComputeSimulation`. Pro reindexaci proměnných reprezentujících stavy automatu opět postačí funkce `Rename` a pro aplikaci existenčního a univerzálního kvantifikátoru je využita funkce `Project`, která na základě predikátu odstraní z BDD nechtěné uzly s danou proměnnou a na jejich potomky aplikuje funktor (`OrApplyFunctor` v případě existenčního kvantifikátoru a `AndApplyFunctor` v případě univerzálního kvantifikátoru). Pro operaci implikace potřebujeme také `ImplicApplyFunctor`. Získání počáteční relace a její iterace odpovídají funkcím *Init* a *Iterate* definovaným v Sekci 4.2.3.

Výsledná simulace je uložena v BDD a k její serializaci slouží funkce `DumpSimulation`, která převede BDD s relací simulace na řetězec vztahů ve formátu `stav <= stav`. Také je možné pomocí metody `MatrixSimulation` transformovat BDD s relací simulace na maticové pole `BinaryRelation`.

Kapitola 6

Vyhodnocení

V této kapitole porovnáváme explicitní reprezentaci konečných/stromových automatů se symbolickou reprezentací konečných automatů, kterou jsme navrhli v Kapitole 4. Byly otestovány tři následující operace: jazykové sjednocení a průnik dvou konečných automatů a výpočet relace simulace nad stavy konečného/stromového automatu.

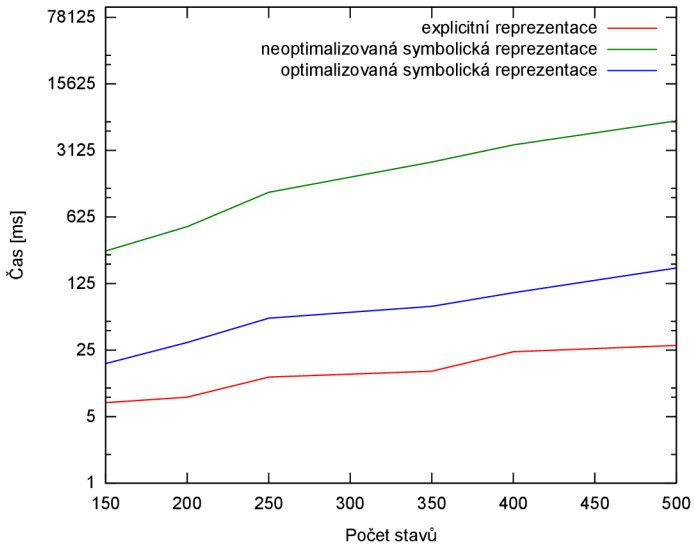
Testy proběhly na stolním počítači pod linuxovou distribucí Mint běžící na virtuálním stroji v 64 b systému Windows 8.1 se čtyřjádrovým procesorem Intel Core 2 Quad Q9400 2,66 GHz a pamětí 8 GB. Testovací množina byla vybrána s ohledem na schopnosti jednotlivých reprezentací.

6.1 Načtení automatu

Čas potřebný k načtení symbolicky reprezentovaného konečného automatu byl změřen a porovnán s explicitní reprezentací, která je již implementovaná v knihovně libVATA. Zároveň jsme zhodnotili zlepšení, které přišlo s optimalizacemi *apply* funktorů pro Booleovu algebru. Jako testovací množinu jsme zvolili konečné automaty nad konstantní abecedou se 130 symboly a různou velikostí (50–500) množiny stavů, které se využívají ve formální verifikaci. Výsledky testování ukazuje Tabulka 6.1 a Obrázek 6.1.

Stavy	**Čas podle typu reprezentace**		
A	**explicitní**	**symbolická**	
		neoptimalizovaná	**optimalizovaná**
150	0,007 s	0,275 s	0,018 s
200	0,008 s	0,494 s	0,030 s
250	0,013 s	1,136 s	0,054 s
350	0,015 s	2,361 s	0,072 s
400	0,024 s	3,573 s	0,100 s
500	0,028 s	6,403 s	0,182 s

Tabulka 6.1: Výsledky testů při načtení automatu



Obrázek 6.1: Výkon při načtení automatu (POZOR: osa y má logaritmickou stupnici!)

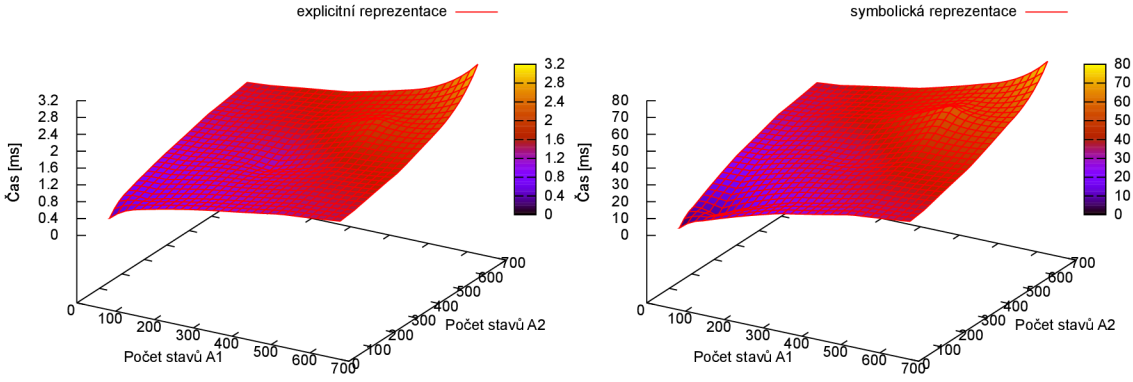
Explicitní reprezentace je podle očekávání výrazně rychlejší, protože symbolická reprezentace vyžaduje náročnější inicializaci BDD. Nejedná se ale o problém, neboť v praxi se nenačítají celé automaty ze souboru, ale jsou postupně sestaveny sadou příkazů pro přímou manipulaci s knihovnou.

6.2 Jazykové sjednocení

Výkon operace jazykového sjednocení dvou symbolicky reprezentovaných konečných automatů byl také změřen a porovnán s explicitní reprezentací. Čas prováděné operace nezahrnuje dobu potřebnou pro načtení automatu. Testovací množina konečných automatů s různou velikostí množin stavů a konstantní abecedou je podobná jako u srovnání výkonnosti načtení konečného automatu v Sekci 6.1. Tabulka 6.2 obsahuje některé vybrané hodnoty a na Obrázku 6.2 je doba trvání u explicitní a symbolické reprezentace.

Stavy		**Čas u reprezentace**	
A_1	A_2	**explicitní**	**symbolická**
53	54	0,0003 s	0,0024 s
111	117	0,0007 s	0,0064 s
312	315	0,0011 s	0,0320 s
493	494	0,0028 s	0,0605 s
646	667	0,0037 s	0,0781 s

Tabulka 6.2: Výsledky testů při jazykovém sjednocení



Obrázek 6.2: Výkon při jazykovém sjednocení

Explicitní reprezentace se opět ukázala rychlejší, ale rozdíl mezi reprezentacemi je malý. Další optimalizace algoritmu jazykového sjednocení by mohly vést k přiblížení se výsledkům, jakých explicitní reprezentace dosahuje.

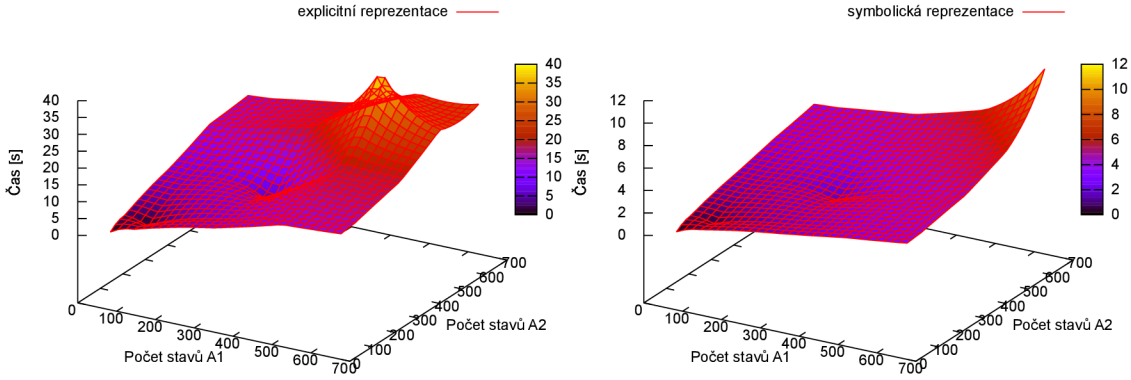
6.3 Jazykový průnik

Jazykový průnik dvou symbolicky reprezentovaných konečných automatů byl opět experimentálně porovnán s explicitní reprezentací. Čas prováděné operace nezahrnuje dobu potřebnou pro načtení automatu. Testované automaty s různým počtem stavů a stejným počtem symbolů zůstávají stejné jako u jazykového sjednocení. V Tabulce 6.3 a na Obrázku 6.3 jsou výsledky testů explicitní a symbolické reprezentace.

Stavy		**Čas u reprezentace**	
A_1	A_2	**explicitní**	**symbolická**
53	54	0,005 s	0,021 s
111	117	0,189 s	0,335 s
312	315	3,496 s	1,344 s
493	494	40,899 s	5,530 s
646	667	26,340 s	11,037 s

Tabulka 6.3: Výsledky testů při jazykovém průniku

Se vzrůstající velikostí množiny stavů konečného automatu se u explicitní reprezentace výrazně prodlužuje doba trvání operace, zatímco symbolická reprezentace se v tomto případě naopak ukazuje jako mnohem efektivnější metoda. Další optimalizace algoritmu jazykového průniku zde sice nejsou prioritou, ale stále je zde prostor pro lepší výsledky.



Obrázek 6.3: Výkon při jazykovém průniku

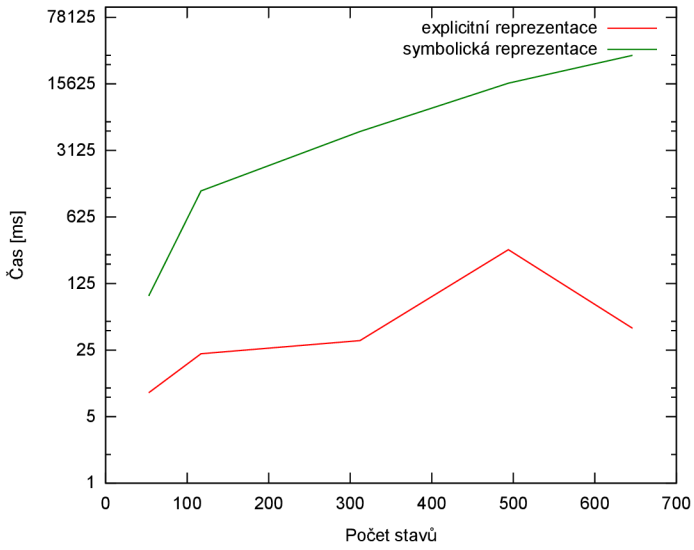
6.4 Relace simulace

Výpočet relace simulace nad stavy symbolicky reprezentovaného konečného automatu byl změřen a porovnán s explicitní reprezentací stromových automatů, která je také implementována v knihovně libVATA, protože daná operace není u explicitní reprezentace konečných automatů dostupná. Testy se tedy mírně liší, neboť stromové automaty chápou množinu počátečních a koncových stavů vzhledem ke konečným automatům opačně, ale pro potřeby srovnání dvou reprezentací jsou postačující. Čas prováděné operace nezahrnuje čas potřebný k načtení automatu. Testovací množina automatů s různou velikostí množiny stavů a konstantní abecedou se od předchozích sekcí nemění. V Tabulce 6.4 jsou výsledky obou reprezentací a jejich grafické znázornění je vidět na Obrázku 6.4.

Stavy	**Čas u reprezentace**	
A	**explicitní**	**symbolická**
53	0,009 s	0,093 s
117	0,023 s	1,174 s
312	0,032 s	4,944 s
494	0,283 s	15,896 s
646	0,042 s	31,176 s

Tabulka 6.4: Výsledky testů při výpočtu relace simulace

Symbolická reprezentace při výpočtu relace simulace výrazně zaostává za explicitní reprezentací, protože používá poměrně náročný algoritmus. Jeho optimalizace jsou na místě a pokud to bude možné, chceme rozdíl maximálně vyrovnat.



Obrázek 6.4: Výkon při výpočtu relace simulace (POZOR: osa y má logaritmickou stupnici!)

6.5 Diskuze výsledků

Výsledky testování prokázaly, že symbolická reprezentace má svůj potenciál jako alternativa explicitní reprezentace, zvláště u komplexních konečných automatů s velkým množstvím stavů a symbolů, ale stále vyžaduje optimalizaci implementovaných algoritmů. Zatímco výkonnost operace jazykového sjednocení a výpočtu relace simulace nepůsobí příliš přesvědčivě, operace jazykového průniku se ukázala jako velmi slibná.

Kapitola 7

Závěr

Cílem této práce bylo navrhnout symbolickou reprezentaci schopnou efektivně pojmout strukturu nedeterministických konečných automatů a implementovat některé jazykové operace, jmenovitě sjednocení, průnik a výpočet relace simulace.

Byly prostudovány a popsány oblasti související s konečnými automaty a BDD a dále byly prozkoumány existující knihovny pro práci s konečnými automaty. V práci je navržena struktura nedeterministických konečných automatů se symbolicky reprezentovanou přechodovou relací, množinou počátečních a koncových stavů. Také byly vypracovány algoritmy výše zmíněných operací. Tato práce rozšiřuje knihovnu libVATA o podporu symbolické reprezentace se zmíněnými operacemi. Výsledky experimentů ukázaly, že symbolická reprezentace má své místo vedle ostatních reprezentací, které knihovna podporuje.

V budoucnosti se chceme soustředit na implementaci dalších operací specifických pro formální verifikaci, například redukci počtu stavů, jazykovou inkluzi a ekvivalenci. Také máme v plánu optimalizovat současné algoritmy a zvýšit jejich efektivitu.

Literatura

[1] Šlapal, J.: *Matematická logika: Základy matematické logiky* [online]. `http://www.math.fme.vutbr.cz/download.aspx?id_file=3582`, 2013.

[2] Vojnar, T.: *Formal Analysis and Verification: Binary Decision Diagrams*. 2013.

[3] Jackson, P.: *Automated Reasoning: Operations on BDDs* [online]. `http://www.inf.ed.ac.uk/teaching/courses/ar/slides/bdd-ops.pdf`, 2013.

[4] Hruška, M.: *Efficient Algorithms for Finite Automata*. Bakalářská práce, FIT VUT, 2013.

[5] Češka, M.; Vojnar, T.; Smrčka, A.: *Teoretická informatika: studijní opora* [online]. `http://www.fit.vutbr.cz/study/courses/TIN/public/Texty/oporaTIN.pdf`, 2013.

[6] van Noord, G.: FSA utilities: A toolbox to manipulate finite-state automata. In *Automata Implementation*, *Lecture Notes in Computer Science*, ročník 1260, editace D. Raymond; D. Wood; S. Yu, Springer Berlin Heidelberg, 1997, ISBN 978-3-540-69205-8, ISSN 0302-9743, s. 87–108.

[7] Klarlund, N.; Møller, A.; Schwartzbach, M. I.: MONA Implementation Secrets. In *Implementation and Application of Automata*, *Lecture Notes in Computer Science*, ročník 2088, editace S. Yu; A. Păun, Springer Berlin Heidelberg, 2001, ISBN 978-3-540-44674-3, ISSN 0302-9743, s. 182–194.

[8] Genet, T.; Tong, V. V. T.: Reachability Analysis of Term Rewriting Systems with Timbuk. In *Logic for Programming, Artificial Intelligence, and Reasoning*, *Lecture Notes in Computer Science*, ročník 2250, editace R. Nieuwenhuis; A. Voronkov, Springer Berlin Heidelberg, 2001, ISBN 978-3-540-45653-7, ISSN 0302-9743, s. 695–706.

[9] Boigelot, B.: *The LASH toolset* [online]. `http://www.montefiore.ulg.ac.be/ boigelot/research/lash/`.

[10] Lengál, O.; Šimáček, J.; Vojnar, T.: VATA: A Library for Efficient Manipulation of Non-deterministic Tree Automata. In *Tools and Algorithms for the Construction and Analysis of Systems*, *Lecture Notes in Computer Science*, ročník 7214, editace C. Flanagan; B. König, Springer Berlin Heidelberg, 2012, ISBN 978-3-642-28756-5, ISSN 0302-9743, s. 79–94.

[11] Holík, L.; Šimáček, J.: *Tool for Computing Simulations* [online]. `http://www.fit.vutbr.cz/research/groups/verifit/tools/sa/`, 2011.

Dodatek A

Obsah CD

Datové médium obsahuje elektronickou verzi technické zprávy a související zdrojové kódy.

A.1 Struktura CD

- `./projekt.pdf` : písemná zpráva
- `./latex/` : zdrojový tvar písemné zprávy
- `./libvata/` : knihovna libVATA se zdrojovými kódy
- `./libvata/README` : informace ke kompilaci knihovny