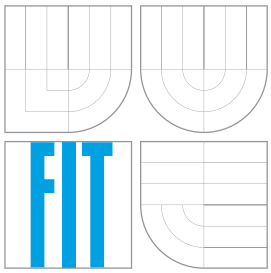
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MULTIDIMENSIONÁLNÍ JAZYKY A JEJICH AUTOMATY

MULTI-DIMENSIONAL LANGUAGES AND THEIR AUTOMATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

LUKÁŠ DIBĎÁK

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. RNDr. ALEXANDER MEDUNA, CSc.

BRNO 2016

Zadání bakalářské práce/18881/2015/xdibda00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Dibďák Lukáš**
Obor: Informační technologie
Téma: **Multidimensionální jazyky a jejich automaty**
Multi-Dimensional Languages and Their Automata
Kategorie: Teoretická informatika

Pokyny:

1. Seznamte se s teorií n-dimensionálních jazyků a automatů, které je přijímají.
2. Zaveďte nový typ n-dimensionálních deterministických konečných automatů dle pokynů vedoucího.
3. Studujte vlastnosti automatů z bodu 2.
4. Aplikujte a implementujte automaty z bodu 2 dle pokynů vedoucího.
5. Diskutujte další vývoj projektu.

Literatura:

- Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, Volume 1-3, Springer, 1997, ISBN 3-540-60649-1
- Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools (2nd Edition), Pearson Education, 2006, ISBN 0-321-48681-1
- Dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Meduna Alexander, prof. RNDr., CSc.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Práce seznamuje s teorií formálních jazyků a konečných automatů. Popisuje zobecnění této teorie do dvou rozměrů. Představuje základní typy dvoudimensionálních automatů, především teselační automaty. Pro teselační automaty jsou nabídnuty algoritmy k jejich determinizaci. Jeden z algoritmů je následně používán přiloženou aplikací pro determinizaci.

Abstract

The Bachelor's Thesis introduces the theory of formal languages and finite automata. It describes generalisation of one-dimensional theory into two dimensions. It introduces basic types of two-dimensional automata, especially on-line tessellation automata. This paper offers algorithms for the process of determinization of on-line tessellation automata. One of the algorithms is used in enclosed application.

Klíčová slova

Konečné automaty, Determinizace, Formální jazyky, Dvoudimensionální jazyky, Dvoudimensionální automaty, Teselační automaty

Keywords

Finite automata, Determinization, Formal languages, Two-dimensional languages, Two-dimensional automata, On-line tessellation automata

Citace

DIBĎÁK, Lukáš. *Multidimensionální jazyky a jejich automaty.* Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Meduna Alexander.

Multidimensionální jazyky a jejich automaty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. RNDr. Alexandera Meduny, CSc. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Lukáš Dibďák
15. května 2016

Poděkování

Chtěl bych poděkovat vedoucímu této práce, prof. RNDr. Alexanderu Medunovi, CSc., za jeho čas, rady a celkovou spolupráci.

Obsah

1 Úvod **3**

2 Jednorozměrné jazyky a konečné automaty **4**
2.1 Abecedy, řetězce a jazyky . . . 4
2.1.1 Abeceda . . . 4
2.1.2 Řetězec a řetězcové operace . . . 4
2.1.3 Mocnění abecedy a vytváření řetězců . . . 5
2.1.4 Jazyk . . . 6
2.2 Regulární výrazy a jazyky . . . 7
2.3 Konečné automaty . . . 8
2.3.1 Definice a chování konečného automatu . . . 9
2.3.2 Proces determinizace automatu . . . 11
2.3.3 Další typy konečných automatů . . . 18

3 Dvourozměrné jazyky a konečné automaty **20**
3.1 Zobecnění principů pro dva rozměry . . . 20
3.2 Regulární výrazy a jazyky . . . 23
3.3 Dvourozměrné konečné automaty . . . 25
3.3.1 Čtyřcestný automat . . . 25
3.3.2 Teselační automat . . . 26

4 Determinizace teselačního automatu **28**
4.1 Rozbor nedeterminismu . . . 28
4.2 Algoritmy determinizace . . . 31
4.2.1 Algoritmus s nedostupnými stavy . . . 31
4.2.2 Algoritmus pouze s dostupnými stavy a funkcemi . . . 33

5 Aplikace **36**
5.1 Uživatelské rozhraní . . . 36
5.1.1 Stavy . . . 36
5.1.2 Vstupní symboly . . . 38
5.1.3 Transitivní funkce . . . 39
5.2 Implementace . . . 40
5.2.1 Implementace vnitřní logiky . . . 41
5.2.2 Implementace grafického rozhraní . . . 42

6 Závěr **44**

Literatura **45**

Přílohy **46**

Seznam příloh 47

A Obsah DVD **48**

Kapitola 1

Úvod

Cílem práce je seznámit s teorií formálních jazyků a konečných automatů v jedné i dvou dimensích. Jádrem práce je představení algoritmů pro determinizaci teselačního automatu a implementace jednoho z algoritmů do přiložené aplikace pro možnost provedení procesu determinizace teselačního automatu.

Práce poskytuje informace o teorii formálních jazyků. Popisuje základní pojmy jednorozměrných jazyků a vysvětluje elementární problematiku těchto jazyků. Informuje o modelech pro charakterizaci regulárních jazyků — regulárních výrazech a konečných automatech. Zmiňuje možnost ekvivalentního převodu mezi těmito modely. U regulárních výrazů definuje, které regulární jazyky značí. Definuje konečné automaty a princip jejich činnosti. Uvádí problematiku determinizace u jednorozměrných automatů a vyobrazuje základní typy konečných automatů.

Po seznámení s teorií jednorozměrných formálních jazyků je poskytnut pohled na zobecnění této teorie do dvou rozměrů. Jsou definovány základní pojmy dvoudimensionálních jazyků. Práce dále představuje vztahy mezi atomickými jazyky a regulárními výrazy, které společně značí regulární jazyky ve dvou rozměrech. Jsou definovány dvoudimensionální automaty, jejich chování a jsou vyobrazeny jejich základní typy.

Pro teselační automat, který patří do skupiny celulárních automatů, jsou rozebrány možné způsoby nedeterminismu. Je vysvětleno, kde tento nedeterminismus vzniká a jsou vylíčena řešení, jak problém nedeterminismu řešit. Jsou nabídnuty algoritmy pro implementaci procesu determinizace teselačního automatu. Problém nedeterminismu a jeho řešení pomocí algoritmů je demonstrován na příkladu nedeterministického teselačního automatu.

Podle přiloženého algoritmu je implementována aplikace umožňující využít procesu determinizace nad teselačním automatem. Tato aplikace je implementována v jazyku Java. Aplikace nabízí přehledné a pro obsluhu jednoduché grafické uživatelské rozhraní. Je popsána implementace vnitřní logiky aplikace i využití grafických tříd, které nabízí jazyk Java, pro implementaci grafického uživatelského rozhraní.

Kapitola 2

Jednorozměrné jazyky a konečné automaty

Kapitola se zabývá vysvětlením základních pojmů z oblasti teorie jednorozměrných jazyků a konečných automatů potřebných k pochopení další teorie dvoudimensionálních jazyků a automatů.

Obsah této kapitoly je náplní předmětu IFJ v rámci bakalářského studia na FIT VUT v Brně, vyučovaného prof. Medunou. Definice a algoritmy zmíněné v této kapitole jsou čerpány z přednáškových materiálů k tomuto předmětu nebo z knihy prof. Meduny [3].

2.1 Abecedy, řetězce a jazyky

Podle analogie jsou základními prvky jazyka písmena jeho abecedy. Z těchto písmen se skládáním vytváří slova. Nekteré z těchto slov tvoří slovo jazyka a některé nikoliv. Jazyk je následně určen množinou slov tohoto jazyka. Tvorba těchto slov je definována pomocí gramatických pravidel, které určují, které posloupnosti písmen tvoří slovo jazyka [6].

2.1.1 Abeceda

Abecedu lze definovat jako libovolnou konečnou neprázdnou množinu symbolů. Podle konvence bývá označována písmenem Σ. Pokud element a náleží do abecedy Σ, pak je tento vztah zapsán jako $a \in \Sigma$ a element a lze prohlásit prvkem abecedy Σ. Níže je přiložena definice abecedy a několik příkladů abeced.

Definice 1. Abeceda je konečná, neprázdná množina elementů, které nazýváme *symboly*.

Příklad 1. $\Sigma = \{0, 1\}$, pak Σ značí binární abecedu [1].

Příklad 2. $\Sigma = \{a, b, c, ..., z\}$, pak Σ značí abecedu malých písmen [1].

Příklad 3. $\Sigma = \{+, -, *, /\}$, pak Σ značí abecedu základních matematických operací.

2.1.2 Řetězec a řetězcové operace

Řetězec, v některých literaturách označován také jako *slovo*, lze definovat jako konečnou posloupnost symbolů nějaké abecedy Σ. Tyto symboly se samozřejmě mohou v posloupnosti opakovat.

Speciálním případem je *prázdný řetězec* nebo-li také *prázdné slovo*. V tomto řetězci se nevyskytuje žádný symbol z abecedy Σ a je podle konvence značen jako ε.

Definice 2. Nechť Σ je abeceda, pak

1. ε je řetězec nad abecedou Σ
2. pokud x je řetězec nad Σ a $a \in \Sigma$, potom xa je řetězec nad abecedou Σ

Nad každým řetězcem lze provést operaci zjištění délky řetězce. Délka řetězce je definována jako počet výskytů kteréhokoliv symbolu abecedy Σ v daném řetězci. Délka řetězce 101 nad abecedou Σ je značena jako $|101|$.

Délka prázdného řetězce ε je nula. Jedná se o jediný řetězec s touto délkou.

Příklad 4. Nechť $\Sigma = \{0, 1\}$, pak řetězce

$$0100,\ 10,\ 0,\ 0101,\ 1111$$

jsou řetězce nad touto abecedou a jejich délky odpovídají

$$|0100| = 4,\ |10| = 2,\ |0| = 1 \text{ a } |\varepsilon| = 0.$$

2.1.3 Mocnění abecedy a vytváření řetězců

Abecedu je možné *mocnit* a tím vyjádřit určitou délku řetězců z abecedy Σ. Mocnění se provádí pomocí exponencionální notace, kde Σ^k značí podmnožinu všech řetězců o velikosti k z abecedy Σ. Počet těchto řetězců se rovná 2^k. Množina všech podmnožin všech různých délek řetězců nad abecedou Σ, nazývána jako *uzávěr abecedy* Σ, je značena Σ^*.

Množinu Σ^* lze tedy také definovat jako

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup ...$$

Uzávěr abecedy Σ, pomocí konvencí zapsáno jako Σ^*, lze vyloučit o prvek prázdného řetězce ε. Uzávěr Σ^* vyloučený o prvek prázdného řetězce ε je označován Σ^+.

Pro množinu Σ^+ lze pak definovat následudící dva vztahy

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup ...$$

$$\Sigma^* = \Sigma^+ \cup \{\varepsilon\}.$$

Příklad 5. $\Sigma = \{0, 1\}$, pak

$$\Sigma^0 = \{\varepsilon\},$$

$$\Sigma^1 = \{0, 1\},$$

$$\Sigma^2 = \{00, 01, 10, 11\} ...$$

Nad každými dvěma řetězci nad abecedou Σ lze provádět operaci *konkatenance*. Operaci konkatenance řetězců x a y lze zjednodušeně popsat jako vytvoření kopie řetězce x následovaného kopií řetězce y.

Pro konkatenanci dvou řetězců x a ε platí, že výsledkem této konkatenance je řetězec x. Formálně,

$$x\varepsilon = \varepsilon x = x.$$

Níže přikládám úplnou definici konkatenance a definice dvou neméně důležitých operací nad řetězcem, *reverzace* a *mocnění řetězce*.

Definice 3. Nechť x a y jsou dva řetězce nad abecedou Σ. Konkatenace řetězců x a y je řetězec xy.

Definice 4. Nechť x je řetězec nad abecedou Σ. Reverzace řetězce x, $reversal(x)$, je definována:

1. pokud $x = \varepsilon$, pak $reversal(\varepsilon) = \varepsilon$
2. pokud $x = a_1 ... a_n$ pak $reversal(a_1 ... a_n) = a_n ... a_1$ pro $n \geq 1$ a a_i pro všechna $a_i = 1, ..., n$

Definice 5. Nechť x je řetězec nad abecedou Σ. Pro $i \geq 0$, i-tá mocnina řetězce x, x^i, je definována:

1. $x^0 = \varepsilon$
2. pro $i \geq 1 : x^i = xx^{i-1}$

2.1.4 Jazyk

Jazyk nad abecedou Σ je definován jako množina řetězců z uzávěru Σ^*, formálně $L \subseteq \Sigma^*$. Konečný jazyk obsahuje konečný počet řetězců n pro $n \geq 1$, zatímco u nekončeného jazyka je počet těchto řetězců nekonečný.

Každý jazyk nad abecedou Σ obsahuje vždy nejméně dva prvky — prázdnou množinu $\varnothing$ (neobsahuje žádný prvek) a prázdný řetězec $\{\varepsilon\}$ (obsahuje jeden prvek) [3].

Definice 6. Nechť Σ^* značí množinu všech řetězců nad Σ. Každá podmnožina $L \in \Sigma^*$ je jazyk nad Σ.

Příklad 6. Nechť $\Sigma = \{a, b\}$, jestliže jazyk $L \subseteq \Sigma^*$ je definován jako

$$L = \{a^n b^n : 1 \leq n \leq 3\},$$

pak jazyk L obsahuje pouze tyto řetězce:

$$ab, aabb, aaabbb.$$

Nad jazyky lze provádět sadu operací. Stejně jako u řetězců lze využít operaci konkatenance. Tu lze u formálních jazyků definovat

Definice 7. Nechť L_1 a L_2 jsou dva jazyky nad Σ. Konkatenace jazyků L_1 a L_2, L_1L_2, je definována jako

$$L_1L_2 = \{xy : x \in L_1 \text{ a } y \in L_2\}.$$

Následují definice klasických množinových operací — *sjednocení*, *průnik* a *rozdíl množin* — specifikované pro dva jazyky L_1 a L_2.

Definice 8. Nechť L_1 a L_2 jsou dva jazyky nad Σ. Sjednocení jazyků L_1 a L_2, $L_1 \cup L_2$, je definováno:

$$L_1 \cup L_2 = \{x : x \in L_1 \text{ nebo } x \in L_2\}.$$

Definice 9. Nechť L_1 a L_2 jsou dva jazyky nad Σ. Průnik jazyků L_1 a L_2, $L_1 \cap L_2$, je definován:

$$L_1 \cap L_2 = \{x : x \in L_1 \text{ a } x \in L_2\}.$$

Definice 10. Nechť L_1 a L_2 jsou dva jazyky nad Σ. Rozdíl jazyků L_1 a L_2, $L_1 - L_2$, je definován:

$$L_1 - L_2 = \{x : x \in L_1 \text{ a } x \notin L_2\}.$$

S využitím operace reverzace řetězce je možné operaci reverzace aplikovat na jazyk.

Definice 11. Nechť L je jazyk nad abecedou Σ. Reverzace jazyka L, $reverse(L)$, je definována:

$$reverse(L) = \{reverse(x) : x \in L\}.$$

Je optimální zmínit následující tři důležité operace nad formálním jazykem — *doplněk*, *mocnění* a *iteraci jazyka*.

Definice 12. Nechť L je jazyk nad abecedou Σ. Doplněk jazyka L, L^-, je definován jako

$$L^- = \Sigma^* - L.$$

Definice 13. Nechť L je jazyk nad abecedou Σ. Pro $i \geq 0$, i-tá mocnina jazyka L, L^i, je definována:

1. $L^0 = \{\epsilon\}$
2. pro $i \geq 1 : L^i = LL^{i-1}$

Definice 14. Nechť L je jazyk nad abecedou Σ. Iterace jazyka L, L^*, a pozitivní iterace jazyka L, L^+, jsou definovány

$$L^* = \bigcup_{i=0}^{\infty} L^i, \quad L^+ = \bigcup_{i=1}^{\infty} L^i.$$

2.2 Regulární výrazy a jazyky

V dalších kapitolách budou vysvětleny dva základní modely pro charakterizaci *regulárních jazyků* — *regulární výrazy* a *konečné automaty*.

Jelikož všechny řetězce nejsou akceptované jazykem, je vhodné umět tyto řetězce popsat. Akceptované řetězce lze popsat pomocí konečných automatů či regulárních výrazů. Oproti konečným automatům, kterým je vyhrazena následující podkapitola 2.3, regulární výrazy nabízejí možnost vyjádřit akceptované řetězce deklarativně - pomocí výrazu obsahujícího symboly a operátory. Tyto operátory značí elementární operace nad jednotlivými symboly a jsou popsány v tabulce 2.1.

Operace	**Popis**
$(a.b)$	představuje konkatenanci symbolů a a b
$(a + b)$	představuje výběr jednoho ze symbolů a nebo b
(a^*)	představuje iteraci nad symbolem a

Tabulka 2.1: Základní operace regulárních výrazů

Jazyky, které regulární výrazy značí, lze definovat tabulkou 2.2 (regulární výraz r znační jazyk L_r a regulární výraz s značí jazyk L_s).

Regulární výraz	**Regulární jazyk, který značí**
$\varnothing$	$\varnothing$
ε	$\{\varepsilon\}$
a, kde $a \in \Sigma$	$\{a\}$
$(r.s)$	$L = L_r L_s$
$(r + s)$	$L = L_r \cup L_s$
(r^*)	$L = L^*$

Tabulka 2.2: Regulární výrazy a jazyky které značí

Regulární jazyk musí být vždy definován regulárním výrazem.

Definice 15. Nechť L je jazyk. L je regulární jazyk, pokud existuje regulární výraz r, který tento jazyk značí.

Příklad 7. Nechť regulární výraz $r = a^+b^+$ definuje regulární jazyk L. Tento jazyk je pak značen

$$L = \{a^n b^n : n \geq 1\}.$$

Regulární výrazy a konečné automaty jsou mezi sebou návzájem převoditelné. Pro každý konečný automat M existuje regulární výraz r [3].

2.3 Konečné automaty

Konečný automat je jednoduchý výpočetní model. Jeho součástí je *vstupní páska*, *čtecí hlava* a *konečné stavové řízení* [3].

- Vstupní páska je rozdělena na čtverce, každý čtverec obsahuje jeden vstupní symbol [3]. Tato páska se pohybuje pod čtecí hlavou
- Čtecí hlava čte symboly ze vstupní pásky. Symbol, který je pod čtecí hlavou je aktuálně zpracovávaný
- Konečné stavové řízení je reprezentováno konečnou množinou stavů společně s konečnou množinou pravidel [3]. Tyto pravidla slouží pro pohyb mezi jednotlivými stavy

Automat následně provádí sekvenční kroky. Tyto sekvenční kroky jsou definovány pravidly. Pravidla určují jak je současný stav měněn. Automat má definovaný počáteční stav a množinu koncových stavů. Pokud je ze vstupní pásky přečten symbol, páska se posune a pod čtecí hlavou se nastaví symbol následující za právě přečteným symbolem. Konečné stavové řízení následně symbol pod čtecí hlavou zpracuje, v množině pravidel vyhledá vhodné pravidlo a posune se, je-li to možné.

Úkolem automatu je se posunout po přečtení vstupní pásky z počátečního stavu do jednoho z definovaných koncových stavů. Výsledkem jsou dvě tvrzení. Automat akceptuje sekvenci symbolů na pásce, popř. tuto sekvenci odmítá [3].

Automat ze vstupní pásky nemusí přečíst v sekvenčním kroku ani jeden symbol, pak dochází k použití pravidla s *ε-přechodem*, tedy provedení nulové *konfigurace*.

2.3.1 Definice a chování konečného automatu

Definice 16. Konečný automat je pětice: $M = (Q, \Sigma, R, s, F)$, kde

- Q je konečná množina stavů
- Σ je vstupní abeceda
- R je konečná množina pravidel tvaru: $pa \rightarrow q$, kde $p, q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$
- $s \in Q$ je počáteční stav
- $F \subseteq Q$ je množina koncových stavů

Chování automatu je reprezentováno konfigurací χ, skládající se z aktuálního stavu automatu a následnou sekvencí symbolů pro přechod mezi stavy. Formálně,

$$\chi \in Q\Sigma^*.$$

Samotná konfigurace obsahuje sekvenci přechodů. *Přechod* reprezentuje jeden výpočetní krok [3] podle vybraného pravidla z množiny pravidel R. Po provedení přechodu vzniká nová konfigurace.

Nula provedených přechodů z konfigurace χ do konfigurace χ naznačuje, že na vstupní pásce nebyl přečten zádný symbol a bylo použito pravidlo automatu s ε-přechodem.

Definice 17. Nechť χ je konfigurace. Konečný automat M provede nula přechodů z χ do χ; zapisujeme:

$$\chi \vdash^0 [\varepsilon]$$

nebo zjednodušeně

$$\chi \vdash^0 \chi.$$

V opačném případě, kdy byl ze vstupní pásky přečten symbol, či v sekvenčních krocích více symbolů, vzniká sekvence přechodů. Níže přikládám definici pro sekvenci přechodů konfigurací.

Definice 18. Nechť

$$\chi_0, \chi_1, ..., \chi_n$$

je sekvence přechodů konfigurací pro $n \geq 1$ a

$$\chi_{i-1} \vdash \chi_i[r_i], r_i \in R$$

pro všechna $i = 1, ..., n$, což znamená:

$$\chi_0 \vdash \chi_1[r_1] \vdash \chi_2[r_2] ... \vdash \chi_n[r_n].$$

Pak M provede n-přechodů z χ_0 do χ_n; zapisujeme:

$$\chi_0 \vdash \chi_n[r_1 ... r_n]$$

nebo zjednodušeně

$$\chi_0 \vdash^n \chi_n.$$

Příklad 8. Nechť konfigurace χ obsahuje sekvenci $qabb$. Automat definovaný množinou stavů a vstupní abecedou

$$Q = \{q, r, s\}, \quad \Sigma = \{a, b\}$$

má definované dvě pravidla

$$\begin{aligned} qa &\to r \quad [1] \\ rb &\to s \quad [2], \end{aligned}$$

pak je možné pomocí pravidla [1] provést přechod

$$qabb \vdash rbb$$

a následně další přechod podle pravidla [2]

$$rbb \vdash sb$$

nebo lze souhrnně zapsat dva přechody konfigurace χ

$$qabb \vdash^2 sb.$$

Pokud je automat schopen přečíst celou sekvenci přechodů a skončí v jednom z definovaných koncových stavů automatu, pak je přečtený řetězec *přijímaným řetězcem* [3].

Definice 19. Nechť $M = (Q, \Sigma, R, s, F)$ je konečný automat. Jazyk přijímaný konečným automatem $M, L(M)$, je definován:

$$L(M) = \{w : w \in \Sigma^*, sw \vdash^* f, f \in F\}.$$

Konečný automat lze znázornit graficky pomocí stavů a symbolů. Stavy jsou uzavřené v kruhu. Pokud je stav koncový, pak je uzavřen ve dvojitém kruhu. Stavy uzavřené v kruhu jsou spojeny. Tyto spoje znázorňují přechody — jsou označeny symbolem a šipkou definující vstupní a výstupní stav.

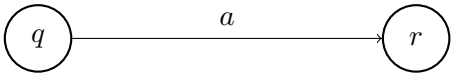
Příklad 9. Nechť $M = (Q, \Sigma, R, s, F)$ je konečný automat obsahující pravidlo

$$qa \vdash r \in R,$$

kde

$$Q = \{s, q, r\}, \quad \Sigma = \{a, b\}.$$

Toto pravidlo lze graficky znázornit pomocí obrázku 2.1.



Obrázek 2.1: Grafické znázornění pravidla automatu

2.3.2 Proces determinizace automatu

Nedeterminizovaný konečný automat může přejít z jedné konfigurace do více dalších. Není tedy specificky určeno do které konfigurace má přejít a tím může vzniknout jiný výsledek než který byl předpokládán. *Determinizovaný konečný automat* smí z jedné konfigurace přejít do maximálně jedné další.

Pro každý konečný automat M existuje *ekvivalentní model* determinizovaného konečného automatu M_d [3]. Proces převedení nedeterministického automatu na ekvivalentní model deterministického automatu se nazývá determinizace automatu. Mezikrokem mezi konečným automatem M a determinizovaným konečným automatem M_d je vytvoření ekvivalentního modelu automatu bez ε-přechodů M'.

Definice 20. Dva modely pro popis formálních jazyků (např. konečné automaty) jsou ekvivalentní, pokud specifikují tentýž jazyk.

Nederminizovanou část konečného automatu lze demonstrovat na příkladu níže.

Příklad 10. Nechť $M = (Q, \Sigma, R, s, F)$ je konečný automat definovaný

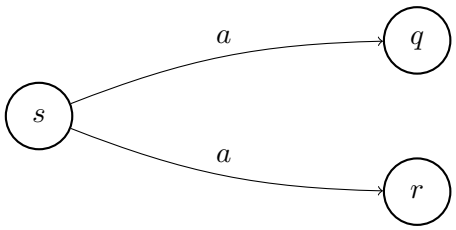
$$Q = \{s, q, r\}, \quad \Sigma = \{a, b\}$$

obsahuje v množině pravidel R mimojiné pravidla, graficky znázorněné na obrázku 2.2

$$sa \vdash q$$
$$sa \vdash r,$$

pak při konfiguraci sab vzniká nejednoznačnost, které pravidlo z pravidel výše použít. Po provedení konfigurace by následující konfigurace mohly být

$$qb$$
$$rb.$$



Obrázek 2.2: Ukázka nedeterminismu konečného automatu M

Pokud konečný automat obsahuje pravidla s ε-přechody, pak smí automat provádět ε-pohyby. Při tomto pohybu automat nečte ze vstupní pásky žádný symbol a hrozí riziko vzniku nedeterminismu. Pro vytvoření determinizovaného automatu je třeba pravidla s těmito přechody odstranit. Řešením může být vytvoření ekvivalentního modelu konečného automatu M', který neobsahuje žádné ε-pravidla.

Při procesu determinizace je nutné nejdříve převést konečný automat M obsahující pravidla s ε-přechody na konečný automat bez pravidel s ε-přechody M'. Níže přikládám definici pro konečný automat bez ε-přechodů.

Definice 21. Nechť $M = (Q, \Sigma, R, s, F)$ je konečný automat. M je konečný automat bez ε-přechodů, pokud pro každé pravidlo

$$pa \rightarrow q \in R,$$

kde $p, q \in Q$, platí:

$$a \in \Sigma \, (a \notin \varepsilon).$$

Pro možnost správného odstranění ε-přechodů je nutné nejdříve pro každý stav automatu vytvořit jeho *ε-uzávěr*. Tento uzávěr musí obsahovat všechny ostatní stavy z automatu do kterých lze ze stavu, pro který je ε-uzávěr vytvářen, přejít bez přečtení vstupního symbolu. Níže je přiložena formální definice pro ε-uzávěr.

Definice 22. Pro každý stav $p \in Q$ je definován ε-uzávěr(p):

$$\varepsilon\text{-uzávěr}(p) = \{q : q \in Q, p \vdash^* q\}.$$

Příklad 11. Nechť $M = (Q, \Sigma, R, s, F)$ je konečný automat, kde

$$Q = \{s, q, r, f\}, \quad \Sigma = \{a\}$$

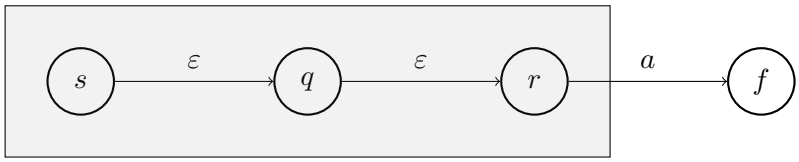
a v množině pravidel R jsou tyto pravidla

$$\begin{aligned} s &\rightarrow q \\ q &\rightarrow r \\ ra &\rightarrow f, \end{aligned}$$

pak ε-uzávěr(s) obsahuje množinu stavů

$$\{s, q, r\},$$

jelikož se do těchto stavů lze dostat bez přečtení symbolu (pouze využitím ε-pravidel. Tento uzávěr je graficky znázorněn na obrázku 2.3.



Obrázek 2.3: Grafické znázornění ε-uzávěru pro stav s

Vytvoření uzávěrů lze zalgoritmizovat. Jeden takový algoritmus 1 je přiložen níže. Vstupem je konečný automat $M = (Q, \Sigma, R, s, F)$. Výstupem ε-uzávěr(p). Jelikož je vytvářen ε-uzávěr pro stav p, předpokládá se, že stav p náleží do množiny stavů automatu M, formálně $p \in Q$.

Následně je potřeba tyto uzávěry zpracovat. Pro každý ε-uzávěr(p), kde $p \in Q$, algoritmus 2 najde všechny pravidla ve tvaru

$$qa \vdash r \, (qa \rightarrow r) \in R,$$

kde

$$a \in \Sigma - \{\varepsilon\}, \quad q \in \varepsilon\text{-uzávěr}(p).$$

Do množiny pravidel bez ε-přechodů R' se přidá pravidlo ve tvaru $pa \vdash r$. Pokud je nějaká shoda mezi stavy ε-uzávěru(p) a koncovými stavy původního automatu M, pak je stav p vložen do koncových stavů automatu bez ε-přechodů M'.

Algoritmus 1 získání hodnot ε-uzávěru stavu p

Vstup: M = (Q, Σ, R, s, F); p ∈ Q
Výstup: ε-uzávěr(p)

1: i := 0; Q_0 := {p};
2: **repeat**
3: i := i + 1;
4: Q_i := Q_{i-1} ∪ {p' : p' ∈ Q, q → p' ∈ R, q ∈ Q_{i-1}};
5: **until** Q_i = Q_{i-1};
6: ε-uzávěr(p) := Q_i.

Algoritmus 2 odstranění ε-přechodů

Vstup: M = (Q, Σ, R, s, F)
Výstup: M' = (Q, Σ, R', s, F')

1: R' := ∅;
2: **for each** p ∈ Q **do begin**
3: R' := R' ∪ {pa → q : p'a → q ∈ R, a ∈ Σ, p' ∈ ε-uzávěr(p), q ∈ Q};
4: **end for**
5: F' := {p : p ∈ Q, ε-uzávěr(p) ∩ F ≠ ∅}.

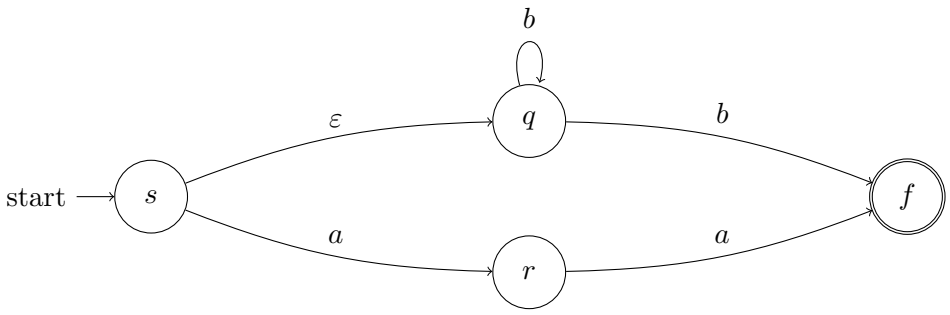
Příklad 12. Nechť $M = (Q, \Sigma, R, s, F)$ je konečný automat s ε-přechody, definovaný

$$Q = \{s, q, r, f\}, \quad \Sigma = \{a, b\},$$

obsahující v množině R pravidla

$$s \to q,\ sa \to r,\ qb \to q,\ qb \to f,\ ra \to f,$$

je vyobrazený na obrázku 2.4.



Obrázek 2.4: Konečný automat obsahující ε-přechod

Pak lze podle algoritmu 1 pro vytvoření ε-uzávěrů vytvořit tyto uzávěry

$$\varepsilon\text{-uzávěr}(s) = \{s, q\}$$
$$\varepsilon\text{-uzávěr}(q) = \{q\}$$
$$\varepsilon\text{-uzávěr}(r) = \{r\}$$
$$\varepsilon\text{-uzávěr}(f) = \{f\}$$

Podle algoritmu 2 je nutné vzniklé ε-uzávěry zpracovat a vytvořit nová pravidla do množiny R' a nové koncové stavy do množiny F' konečného automatu M'. Konečná množina stavů Q, počáteční symbol s a množina vstupních symbolů Σ se duplikuje z konečného automatu s ε-přechody M. Množinu nových pravidel R' prezentuje tabulka 2.3, množinu nových koncových stavů F' pak tabulka 2.4.

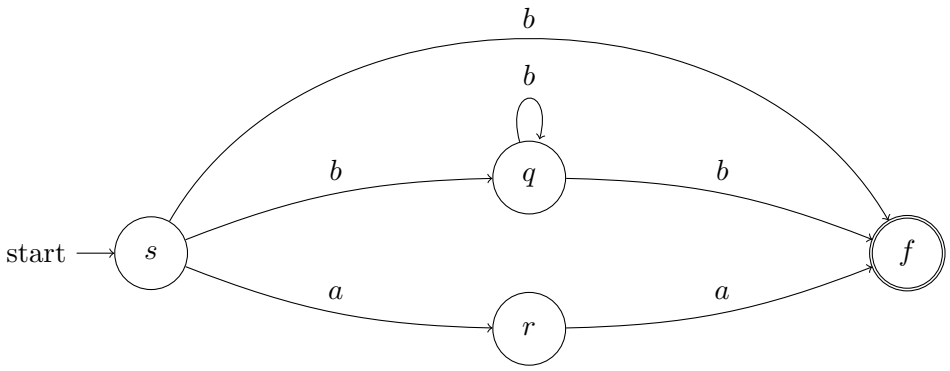
ε-**uzávěr**(s)	ε-**uzávěr**(q)	ε-**uzávěr**(r)	ε-**uzávěr**(f)
$sa \to r$	$qb \to q$	$ra \to f$	
$sb \to q$	$qb \to f$		
$sb \to f$			

Tabulka 2.3: Pravidla automatu bez ε-přechodů

ε-**uzávěr**(s)	ε-**uzávěr**(q)	ε-**uzávěr**(r)	ε-**uzávěr**(f)
			f

Tabulka 2.4: Koncové stavy automatu bez ε-přechodů

Výsledný automat bez ε-přechodů, $M' = (Q, \Sigma, R', s, F')$, je zobrazen na obrázku 2.5.



Obrázek 2.5: Ekvivalentní konečný automat bez ε-přechodů

Druhým krokem v procesu determinizace je odstranění nedeterministických pravidel automatu. Vstupem je konečný automat bez ε-přechodů M, výstupem konečný deterministický automat M_d. V tomto textu budou prezentovány dva algoritmy pro determinizaci konečného automatu.

První algoritmus 3 vytváří stavy ze všech podmnožin množiny stavů vstupního konečného automatu M. Bohužel vytváří nedostupné stavy do kterých automat nemůže vkročit pomocí žádné konfigurace. Pro praktické využití je determinizovaný automat vytvořený tímto algoritmem nevhodný, jelikož obsahuje redundantní data. Nejprve se vytvoří všechny podmnožiny stavů. Těchto kombinací pro n stavů je $2^n - 1$. Vytvoření těchto podmnožin je demonstrováno v příkladu níže.

Příklad 13. Nechť $M = (Q, \Sigma, R, s, F)$ je konečný automat, obsahující v množině stavů tři stavy, formálně zapsáno

$$Q = \{q, r, s\}.$$

Tyto stavy vytvořené podmnožinami Q pro determinizaci jsou

$$\{q\}, \{r\}, \{s\},$$
$$\{q, r\}, \{q, s\}, \{r, s\},$$
$$\{q, r, s\}.$$

Po vytvoření podmnožin se pro každý stav podmnožiny, který náleží podmnožině prohledají pravidla ve tvaru $pa \vdash q \in R$, kde $a \in \Sigma$. Pokud jeden ze stavů podmnožiny odpovídá stavu p, pak se vytvoří pravidlo, kde je stav p nahrazen celou touto podmnožinou. Pokud pro symol a existuje více pravidel vyhovujících pro pa, pak se výstupní stavy q slučují a vytvoří nový stav. Pokud je jeden prvek z této podmnožiny shodný s některým koncovým stavem M, pak je tato podmnožina označena jako koncový stav.

Algoritmus 3 determinizace s vytvářením nedostupných stavů

Vstup: $\mathrm{M} = (\mathrm{Q}, \Sigma, \mathrm{R}, \mathrm{s}, \mathrm{F})$ - bez ε-přechodů
Výstup: $\mathrm{M_d} = (\mathrm{Q_d}, \Sigma, \mathrm{R_d}, \mathrm{s_d}, \mathrm{F_d})$

1: $\mathrm{Q_d} := \{\mathrm{Q'} : \mathrm{Q'} \subseteq \mathrm{Q}, \mathrm{Q'} \neq \varnothing\}$;
2: $\mathrm{R_d} := \varnothing$;
3: **for each** $\mathrm{Q'} \in \mathrm{Q_d}$ **and** $\mathrm{a} \in \Sigma$ **do begin**
4: $\quad \mathrm{Q''} := \{\mathrm{q} : \mathrm{p} \in \mathrm{Q'}, \mathrm{pa} \to \mathrm{q} \in \mathrm{R}\}$;
5: $\quad$ **if** $\mathrm{Q''} \neq \varnothing$ **then** $\mathrm{R_d} := \mathrm{R_d} \cup \{\mathrm{Q'a} \to \mathrm{Q''}\}$;
6: **end for**
7: $\mathrm{s_d} := \{\mathrm{s}\}$;
8: $\mathrm{F_d} := \{\mathrm{F'} : \mathrm{F'} \in \mathrm{Q_d}, \mathrm{F'} \cap \mathrm{F} \neq \varnothing\}$.

Druhý algoritmus 4 postupuje ve všech možných konfiguracích reálného automatu. Vznikají tedy pouze *dostupné stavy* nutné pro přirozené chování automatu. Zároveň nejsou vytvářena žádná redundantní data. Definici dostupného stavu přikládám níže.

Definice 23. Nechť $M = (Q, \Sigma, R, s, F)$ je konečný automat. Stav

$$q \in Q$$

je dostupný, pokud existuje

$$w \in \Sigma^*,$$

pro který platí

$$sw \vdash^* q.$$

Jinak q je nedostupný.

Algoritmus 4 determinizace s vytvářením pouze dostupných stavů

Vstup: $M = (Q, \Sigma, R, s, F)$ - bez ε-přechodů
Výstup: $M_d = (Q_d, \Sigma, R_d, s_d, F_d)$ - bez nedostupných stavů

1: $s_d := \{s\}$; $Q_{new} := \{s_d\}$;
2: $R_d := \varnothing$; $Q_d := \varnothing$; $F_d := \varnothing$;
3: **repeat**
4: $Q' \in Q_{new}$; $Q_{new} := Q_{new} - \{Q'\}$; $Q_d := Q_d \cup \{Q'\}$;
5: **for each** $a \in \Sigma$ **do begin**
6: $Q'' := \{q : p \in Q', pa \to q \in R\}$;
7: **if** $Q'' \neq \varnothing$ **then** $R_d := R_d \cup \{Q'a \to Q''\}$;
8: **if** $Q'' \notin Q_d \cup \{\varnothing\}$ **then** $Q_{new} := Q_{new} \cup \{Q''\}$
9: **end for**
10: **if** $Q' \cap F \neq \varnothing$ **then**
11: $F_d := F_d \cup \{Q'\}$
12: **end if**
13: **until** $Q_{new} = \varnothing$.

Následuje jednoduchá ukázka odstranění nedeteminismu. Avšak, tento princip je samozřejmě využitelný i u složitějších automatů s výšší mírou nedeterminismu.

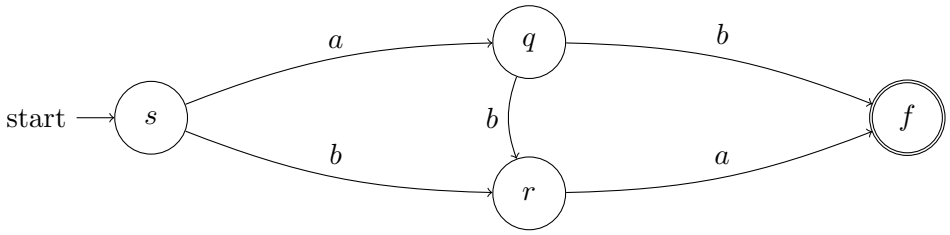
Příklad 14. Nechť $M = (Q, \Sigma, R, s, F)$ je konečný automat neobsahující žádná pravidla s ε-přechody, který je definován

$$Q = \{s, q, r, f\}, \quad \Sigma = \{a, b\},$$

s těmito pravidly v množině pravidel R

$$sa \to q, \quad sb \to r, \quad qb \to r, \quad qb \to f, \quad ra \to f,$$

pak ve stavu q vzniká nedeterminismus, jelikož lze pomocí symbolu b, formálně pomocí konfigurace qb přejít do více než jedné další konfigurace. Konkrétně do r či f. Tento konečný automat M je vyobrazen na obrázku 2.6.



Obrázek 2.6: Nedeterministický konečný automat

V následujícím textu budu demonstrovat odstranění tohoto nedeterminismu bez vytváření nedostupných stavů pomocí algoritmu 4.

V každém průchodu cyklu algoritmu se zpracovává jeden stav automatu z množiny stavů ke zpracování. Při prvním průchodu se zpracovává inicializační stav. Následně se pro každý symbol prohledávají pravidla ve tvaru

$$pa \rightarrow q \in R,$$

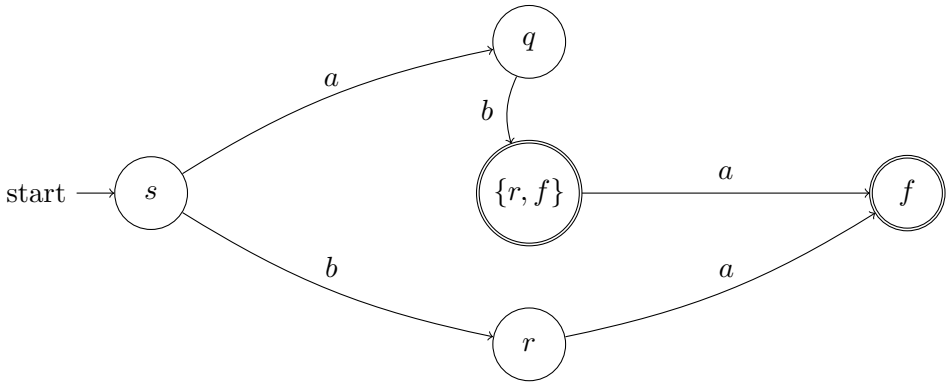
kde p je zpracovávaný stav a a zpracovávaný symbol. Pokud je alespoň jedno takové pravidlo nalezeno, vytvoří se nové pravidlo. Zde může při více výskytech vstupní části pa dojít ke slučování stavů ve výstupní části q. Pokud je ve zpracovávaném stav obsažen jeden z koncových stavů (zpracovávaný stav se může skládat z více stavů z důvodu slučování stavů), pak je zpracovávaný stav vložen mezi koncové stavy nového automatu M_d. Pokud je fronta stavů ke zpracování prázdná, pak algoritmus končí.

Pro tento příklad algoritmus začíná zpracovávat počáteční stav s. Algoritmus nalezne pravidla pro každý ze symbolů, tedy pro vstupní část ve tvaru sa i sb. Vytvoří pravidla a výstupní stavy q, respektive r zařadí do fronty ke zpracování. Dále se zpracovává stav q. Zde dochází k nedeterminismu a pro vstupní část qb jsou nalezeny dva výstupní stavy r a f. Dochází tedy ke sloučení těchto stavů a stav r_f je zařazen do fronty ke zpracování. Podobně algoritmus pokračuje i pro ostatní stavy r, r_f a f. Jelikož už není vytvořen žádný nový stav, který by mohl být zpracován, algoritmus následně končí.

Celý průběh algoritmu je zaznamenán v tabulce 2.5, výsledný automat je graficky zobrazen na obrázku 2.7.

Průchod	**Zpracovávaný stav**	**Vytvořené pravidla**	**Stavy ke zpracování**
1	s	$sa \rightarrow q$, $sb \rightarrow r$	q, r
2	q	$qb \rightarrow r\_f$	r_f
3	r	$ra \rightarrow f$	f
4	r_f	$r\_fa \rightarrow f$	
5	f		

Tabulka 2.5: Průchody algoritmu nedeterminizovaným automatem



Obrázek 2.7: Transformovaný deterministický konečný automat

2.3.3 Další typy konečných automatů

V této kapitole budou stručně vysvětleny další dva typy konečných automatů - *úplný determinizovaný konečný automat* a *dobře specifikovaný konečný automat*.

Úplný determinizovaný konečný automat má definovaný výstupní stav q pro kteroukoliv dvojici vstupní části pravidla pa,

$$pa \rightarrow q \in R,$$

kde

$$p \in Q, \quad a \in \Sigma.$$

Následuje formální definice úplného determinizovaného konečného automatu.

Definice 24. Nechť $M = (Q, \Sigma, R, s, F)$ je determinizovaný konečný automat. M je úplný, pokud pro libovolné

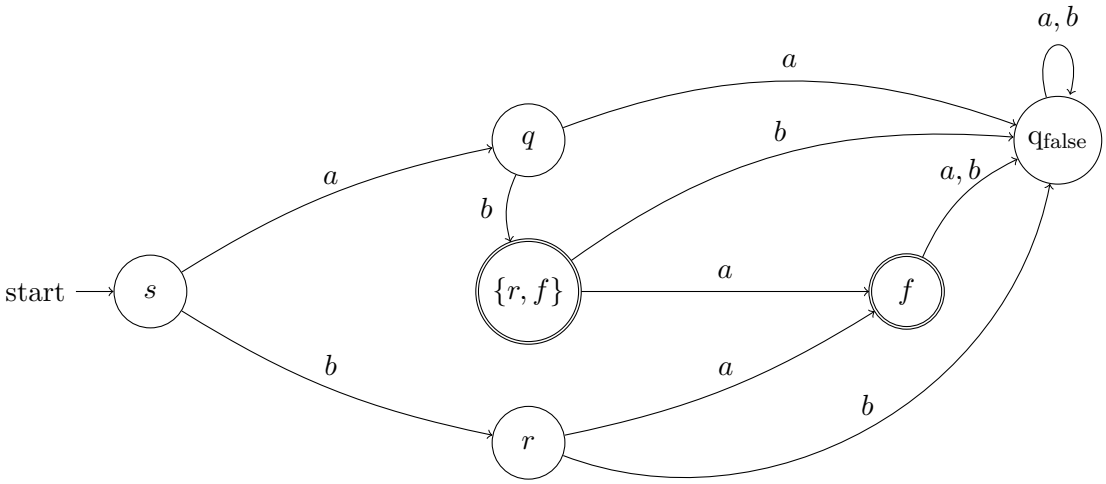
$$p \in Q, \quad a \in \Sigma$$

existuje právě jedno pravidlo

$$pa \rightarrow q \in R$$

pro nějaké $q \in Q$. Jinak M je neúplný.

Příklad úplného deterministického konečného automatu je uveden na obrázku 2.8. Tento automat je upravený podle pravidel pro vytvoření úplného determinizovaného konečného automatu. Základem je determinizovaný konečný automat 2.7 z minulé podkapitoly 2.3.2. Dále přikládám algoritmus 5 podle kterého je možné tento automat vytvořit.



Obrázek 2.8: Úplný deterministický konečný automat

Algoritmus 5 vytvoření úplného determinizovaného konečného automatu

Vstup: $\mathrm{M} = (\mathrm{Q}, \Sigma, \mathrm{R}, \mathrm{s}, \mathrm{F})$
Výstup: $\mathrm{M_c} = (\mathrm{Q_c}, \Sigma, \mathrm{R_c}, \mathrm{s}, \mathrm{F})$

1: $\mathrm{Q_c} := \mathrm{Q} \cup \{\mathrm{q_{false}}\}$;
2: $\mathrm{R_c} := \mathrm{R} \cup \{\mathrm{qa} \to \mathrm{q_{false}} : \mathrm{a} \in \Sigma, \mathrm{q} \in \mathrm{Q_c}, \mathrm{qa} \to \mathrm{p} \notin \mathrm{R}, \mathrm{p} \in \mathrm{Q}\}$.

Posledním typem jednorozměrného konečného automatu, který v této práci bude vysvětlen je dobře specifikovaný konečný automat. Pro vytvoření tohoto automatu je třeba zbavit konečný automat ε-přechodů a provést determinizaci. Tím bude zajištěno, že automat nebude obsahovat nedostupné stavy. Dále je nutné automat zbavit *neukončujících stavů* a převést ho na úplný determinizovaný konečný automat. Níže je přiložena definice ukončujícího stavu.

Definice 25. Nechť $M = (Q, \Sigma, R, s, F)$ je deterministický konečný automat. Stav

$$q \in Q$$

je ukončující, pokud existuje řetězec

$$w \in \Sigma^*,$$

pro který platí:

$$qw \vdash^* f,\ f \in F.$$

Jinak q je neukončující.

Po splnění všech těchto podmínek vzniká dobře specifikovaný konečný automat. Avšak i tento automat obsahuje právě jeden neukončující stav — $\mathrm{q_{false}}$.

Kapitola 3

Dvourozměrné jazyky a konečné automaty

Následující kapitola shrnuje poznatky potřebné k pochopení základních principů dvoudimensionálních jazyků a automatů. Zobecňuje principy jednorozměrných jazyků do dvou rozměrů. Seznamuje s regulárními jazyky ve dvou rozměrech. Představuje základní typy dvoudimensionálních automatů s důrazem na teselační automat, na který je následně navázána kapitola 4 vysvětlující proces determinizace nad tímto automatem.

3.1 Zobecnění principů pro dva rozměry

„Podstatou dvourozměrných jazyků je zobecnění konceptů a technik teorie formálních jazyků do dvou dimenzí“ [2].

Dvourozměrné jazyky zavádí nový pojem, kterým je *obraz*. Jedná se o řetězec zobecněný z jednorozměrných jazyků z kapitoly 2 do dvou rozměrů. Obraz tedy reprezentuje *dvourozměrný řetězec*.

Definice 26. Dvourozměrný řetězec (nebo také obraz) nad abecedou Σ je dvourozměrné pole elementů z abecedy Σ. Množina všech dvourozměrných řetězců nad abecedou Σ je značena Σ^{**}. Dvourozměrný jazyk nad abecedou Σ je podmnožinou Σ^{**}[5].

Nechť existuje obraz p, kde $p \in \Sigma^{**}$, pak $l_1(p)$ značí počet řádků p a $l_2(p)$ značí počet sloupců p. Dvojice

$$l_1(p) \times l_2(p)$$

definuje velikost p. Existuje jediný obraz s velikostí

$$0 \times 0,$$

který je označován jako *prázdný obraz*, Λ [4]. *Obrazy* s velikostí

$$0 \times n,\ n \times 0,$$

kde $n > 0$, nejsou definovány. Množinu všech obrazů o velikosti

$$m \times n,$$

kde $m, n > 0$, nad abecedou Σ označujeme jako $\Sigma^{m \times n}$ [2].

Dále, nechť p je obraz. Jestliže i, j jsou definovány

$$0 \leq i \leq l_1(p)$$
$$0 \leq j \leq l_2(p),$$

pak

$$p(i,j) \text{ nebo-li } p_{i,j}$$

značí symbol na souřadnicích i, j *obrazu* p.

Příklad 15. Nechť $\Sigma = \{a\}$, dvoudimensionální jazyk reprezentovaný množinou všech obrazů obsahujících pouze symbol a, které jsou definovány dvěmi řádky lze být zapsána

$$L = \{p \,|\, p \in \Sigma^{**} \text{ a } l_1(p) = 2\} \text{ [5]}.$$

Graficky lze některé z těchto obrazů znázornit podle obrázku 3.1.

a	a	a
a	a	a

Obrázek 3.1: Ukázka prvních třech obrazů dvoudimensionálního jazyka L

Dvoudimensionální řetězce (obrazy) definují sadu operací. První z této sady operací pro obraz p je operace $\hat{p}$. Jestliže p definuje svojí velikost jako

$$m \times n,$$

pak se velikost $\hat{p}$ rovná

$$(m+2) \times (n+2),$$

kdy po okrajích obrazu p je přidán speciální symbol značící *hranici obrazu* [5]. Tento speciální symbol je identifikován pomocí znaku #, kde # $\notin \Sigma$.

Ukázka operace $\hat{p}$ pro obraz p je vyobrazena na obrázku 3.2.

a	a	a
a	a	a
a	a	a

#	#	#	#	#
#	a	a	a	#
#	a	a	a	#
#	a	a	a	#
#	#	#	#	#

Obrázek 3.2: Obraz p a následné provedení operace $\hat{p}$

Dalšími dvěma operacemi, které budou v tomto textu prezentovány jsou *řádková* a *sloupcová konkatenance* [5]. Nechť p a q jsou dva dvourozměrné řetězce nad abecedou Σ

$$p_{k,l}$$
$$q_{m,n},$$

kde

$$k,\, l,\, m,\, n > 0.$$

Tyto řetězce jsou vyobrazeny na obrázku 3.3.

$$p = \begin{array}{|ccc|}\hline p_{1,1} & \cdots & p_{1,l} \\ \vdots & \ddots & \vdots \\ p_{k,1} & \cdots & p_{k,l} \\ \hline \end{array} \qquad q = \begin{array}{|ccc|}\hline q_{1,1} & \cdots & q_{1,n} \\ \vdots & \ddots & \vdots \\ q_{m,1} & \cdots & q_{m,n} \\ \hline \end{array}$$

Obrázek 3.3: Grafická ilustrace obrazů p a q

Řádková konkatenance nad řetězci p, q je možná pouze pokud

$$l = n$$

a sloupcová konkatenance je možná pouze pokud

$$k = m.$$

Ukázky obou těchto konkatenancí jsou ilustrovány na obrázcích 3.4, respektive 3.5.

$$p \ominus q = \begin{array}{|ccc|}\hline p_{1,1} & \cdots & p_{1,l} \\ \vdots & \ddots & \vdots \\ p_{k,1} & \cdots & p_{k,l} \\ \hline q_{1,1} & \cdots & q_{1,n} \\ \vdots & \ddots & \vdots \\ q_{m,1} & \cdots & q_{m,n} \\ \hline \end{array}$$

Obrázek 3.4: Řádková konkatenance obrazů p a q

$$p \oslash q = \begin{array}{|ccc|ccc|}\hline p_{1,1} & \cdots & p_{1,l} & q_{1,1} & \cdots & q_{1,n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ p_{k,1} & \cdots & p_{k,l} & q_{m,1} & \cdots & q_{m,n} \\ \hline \end{array}$$

Obrázek 3.5: Sloupcová konkatenance obrazů p a q

Jestliže dochází ke konkatenanci libovolného neprázdného obrazu a prázdného obrazu Λ, pak výsledek je vždy definován a Λ se v tomto případě chová jako neutrální prvek [5].

Řádková a sloupcová konkatenance jsou kromě dvoudimensionálních řetězců aplikovatelné i na dva dvoudimensionální jazyky.

Definice 27. Nechť L_1 a L_2 jsou dvoudimensionální jazyky nad abecedou Σ. Řádková konkatenance jazyků L_1 a L_2, $L_1 \ominus L_2$, je definována

$$L_1 \ominus L_2 = \{p \ominus q \mid p \in L_1 \text{ a } q \in L_2\} \text{ [5]}.$$

Definice 28. Nechť L_1 a L_2 jsou dvoudimensionální jazyky nad abecedou Σ. Sloupcová konkatenance jazyků L_1 a L_2, $L_1 \oplus L_2$, je definována

$$L_1 \oplus L_2 = \{p \oplus q \mid p \in L_1 \text{ a } q \in L_2\} \text{ [5]}.$$

Pomocí iterací řádkových nebo sloupcových konkatenancí nad dvoudimensionálním jazykem lze získat uzávěr dané konkatenance nad daným dvoudimensionálním jazykem [5].

Definice 29. Nechť L je dvourozměrný jazyk. Sloupcový uzávěr L je definován

$$L^{*\oplus} = \bigcup_{i=0}^{\infty} L^{i\oplus}$$

$$L^{0\oplus} = \Lambda, \quad L^{1\oplus} = L, \quad L^{*\oplus} = L \oplus L^{(n-1)\oplus}.$$

Definice 30. Nechť L je dvourozměrný jazyk. Řádkový uzávěr L je definován

$$L^{*\ominus} = \bigcup_{i=0}^{\infty} L^{i\ominus}$$

$$L^{0\ominus} = \Lambda, \quad L^{1\ominus} = L, \quad L^{*\ominus} = L \ominus L^{(n-1)\ominus}.$$

Příklad 16. Nechť L je jazyk nad abecedou Σ. Pak L^{**} lze definovat pomocí konkatenancí

$$L^{**} = (L^{*\oplus})^{*\ominus}.$$

3.2 Regulární výrazy a jazyky

Nechť Σ je libovolná abeceda, pak prázdný jazyk, $\varnothing$ a každý jazyk $\{\boxed{a}\}$, kde $a \in \Sigma$, jsou nazývány *atomické jazyky* nad abecedou Σ [5].

Množina regulárních operací, které lze nad atomickými jazyky aplikovat, je

$$\mathcal{R} = \{\ominus,\ \oplus,\ *\ominus,\ *\oplus,\ \cup,\ \cap,\ ^c\}.$$

„Jazyk nad abecedou Σ je regulární, pokud ho lze získat z nějakého atomického jazyka pomocí konečně mnoha aplikací operací z $\mathcal{R}$. Regulární výraz je pak předpis, který udává, jakým způsobem je daný jazyk pomocí regulárních operací z atomických jazyků získán [2].“

Které operace jednotlivé operátory z množiny $\mathcal{R}$ značí je uvedeno v tabulce 3.1.

Definice 31. Regulární výraz nad abecedou Σ je rekurzivně definován

1. $\varnothing$ a každé $a \in \Sigma$ jsou regulární výrazy
2. Nechť α a β jsou regulární výrazy, pak $(\alpha) \cup (\beta)$, $(\alpha) \cap (\beta)$, $^c(\alpha)$, $(\alpha) \ominus (\beta)$, $(\alpha) \oplus (\beta)$, $(\alpha)^{*\ominus}$, $(\alpha)^{*\oplus}$ jsou regulární výrazy [5]

Každý regulární výraz nad abecedou Σ značí dvoudimensionální jazyk nad abecedou Σ.

Definice 32. Dvoudimensionální jazyk $L \subseteq \Sigma^{**}$ je regulární, jestliže existuje regulární výraz nad Σ, který jej značí.

Operátor	**Význam operátoru**
$(\alpha) \cup (\beta)$	značí sjednocení jazyků α a β
$(\alpha) \cap (\beta)$	značí průnik jazyků α a β
${}^{c}(\alpha)$	doplněk jazyka α
$(\alpha) \ominus (\beta)$	značí řádkovou konkatenanci α a β
$(\alpha) \oplus (\beta)$	značí sloupcovou konkatenanci α a β
$(\alpha)^{*\ominus}$	značí řádkovou iteraci α
$(\alpha)^{*\oplus}$	značí sloupcovou iteraci α

Tabulka 3.1: Operátory nad regulárními výrazy [2]

Příklad 17. Nechť $\Sigma = \{a, b\}$, pak regulární výraz

$$(((a \ominus b)^{*\ominus}) \oplus ((b \ominus a)^{*\ominus}))^{*\oplus}$$

značí jazyky vzhledu "šachovnice" se sudou délkou stran.

Dva takové příklady obrazů znázorňuje obrázek 3.6.

a	b	a	b	a	b
b	a	b	a	b	a
a	b	a	b	a	b
b	a	b	a	b	a

a	b
b	a

Obrázek 3.6: Graficky znázorněné obrazy vyhovující regulárnímu výrazu

3.3 Dvourozměrné konečné automaty

Tato podkapitola popisuje dva základní typy dvoudimensionálních konečných automatů, konkrétně *čtyřcestný* a *teselační*. Teselačnímu automatu je věnována větší pozornost vzhledem k jeho determinizaci v kapitole 4.

Dvoudimensionální konečný automat čte dvourozměrnou pásku a na základě vnitřních pravidel, které jsou nazývány *transitivní funkce*, se po této pásce pohybuje. Podobně jako jednorozměrný automat, dvourozměrný automat iniciuje svojí činnost inicializačním stavem a postupně zpracovává vstupní pásku. Oba tyto typy automatů obsahují akceptující i odmítající stav (stavy) pro přijmutí či odmítnutí daného obrazu.

3.3.1 Čtyřcestný automat

Dvoudimensionální automat, rozšiřující klasický dvoudimensionální automat, rozeznává řetězce pohybem ve čtyřech směrech — vlevo, vpravo, nahoru a dolů.

Definice 33. Nedeterministický (deterministický) čtyřcestný konečný automat, značený jako 4NFA (4DFA), je sedmice $\mathcal{A} = (\Sigma, Q, \triangle, q_0, q_a, q_r, \delta)$ kde

- Σ je vstupní abeceda
- Q je konečná množina stavů
- $\triangle = \{R, L, U, D\}$ je množina směrů pro pohyb automatu
- q_0 je inicializační stav
- $q_a, q_r \in Q$ je "akceptující", respektive "odmítající" stav
- $\delta : \{q_a, q_r\} \times \Sigma \to 2^{Q \times \triangle}$ ($\delta : \{q_a, q_r\} \times \Sigma \to Q \times \triangle$) je transitivní funkce [5].

Jedná se o model konečného stavového řízení pomocí Q, pro jednorozměrné automaty popsáno v kapitole 2.3, který čte vstupní obraz. Pohyb čtecí hlavy je ovlivňován transitivní funkcí, která rozhoduje kam se automat dále na vstupním obraze bude pohybovat [2].

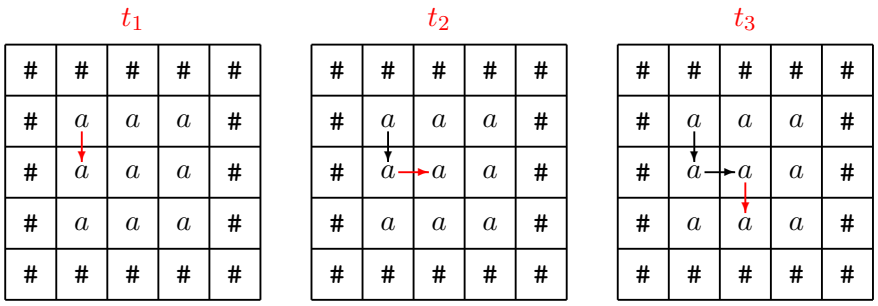
U tohoto typu automatu je transitivní funkci předán aktuální stav, aktuální symbol pozice na které se nachází ve vstupním obrazu a výstupem transitivní funkce je nový stav pro konečné stavové řízení a směr, kterým se na vstupním obrazu čtecí hlava bude pohybovat.

Pokud automat narazí na stav q_a nebo q_r, pak automat zastaví, protože pro tyto stavy, podle definice čtyřcestného konečného automatu, nejsou definovány žádné transitivní funkce, které by pro další pohyb automatu mohly být použity.

Jestliže automat narazil na stav q_a, pak je tento vstupní obraz přijat. Pokud narazil na stav q_r, pak je obraz odmítnut.

Automat pracuje nad vstupním obrazem p, nad kterým byla provedena operace $\hat{p}$ prezentována v podkapitole 3.1. Pak automat ví, kdy se nachází na okraji vstupního obrazu a automaticky se vrátí zpět do p v následujícím kroku.

Ukázkové kroky čtyřcestného automatu jsou graficky vyobrazeny na obrázku 3.7.



Obrázek 3.7: Ukázka pohybu čtyřcestného automatu

3.3.2 Teselační automat

Předchozí automat se pohyboval po vstupním obrazu sekvenčně, kdy při každém kroku zpracoval pouze jeden symbol a podle tohoto symbolu se na vstupní pásce přesunul v jednom směru. Teselační automat patří do skupiny *celulárních automatů*. U celulárních automatů lze vstupní pásku chápat jako pole buňek, kde každá tato buňka v každém kroku může měnit svůj stav podle svého okolí. Kroky, které celulární automat provádí jsou *diskrétní*. Narozdíl od čtyřcestného automatu je možné zpracovávat více buňek v jednom kroku.

Jak je zmíněno výše, teselační automat patří do skupiny celulárních automatů. Provádí tedy diskrétní kroky, avšak při každém kroku nemění hodnoty všech buňek. Teselační automat prochází vstupní pásku diagonálně a podle předchozí zpracované diagonály, která je částečným okolím aktuální diagonály, asociuje všechny stavy aktuálně zpracovávané diagonály současně. Každá buňka má ve svém okolí z předchozí diagonály dva zpracovávané stavy, stav "nad buňkou"a stav "vlevo od buňky". Následně je této buňce přiřazen stav podle transitivní funkce, která je zobrazena v definici níže [5].

Definice 34. Nedeterministický (deterministický) dvoudimensionální teselační automat, značený jako 2OTA (2-DOTA), je pětice $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ kde

- Σ je vstupní abeceda
- Q je konečná množina stavů
- $q_0 \in Q$ je inicializační stav,
- $F \in Q$ je množina akceptujících stavů
- $\delta : Q \times Q \times \Sigma \to 2^Q$ ($\delta : Q \times Q \times \Sigma \to Q$) je transitivní funkce

Jelikož se, narozdíl od čtyřcestného automatu, teselační automat nemůže libovolněkrát vrátit na některou z buněk, celý běh automatu $\mathcal{A}$ na obrazu $\hat{p}$ má konstantní počet kroků. Těchto kroků je

$$l_1(p) + l_2(p) - 1.$$

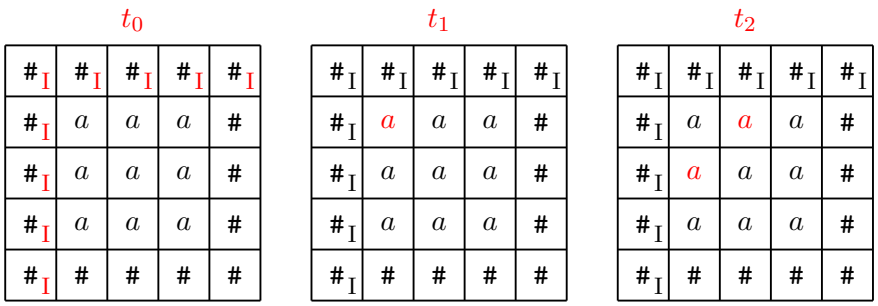
V čase $t = 0$ jsou všechny pozice prvního řádku a sloupce $\hat{p}$ asociovány inicializačním stavem.

V čase $t = 1$ je přečtena první buňka obrazu p, $p(1,1)$, a je jí přiřazen stav podle inicializační transitivní funkce ve tvaru $\delta(q_0, q_0, p(1,1))$.

V čase $t = 2$ jsou zpracovány buňky na pozici $(1, 2)$ a $(2, 1)$ a je jim přiřazen stav podle odpovídajících transitivních funkcí. Automat dále pokračuje po všech diagonálách a zpracovává tímto způsobem všechny stavy v diagonále.

V čase $t = k$, kde $k = i + j - 1$, jsou současně asociovány všechny stavy pro pozice (i, j). Automat $\mathcal{A}$ rozeznává a přijímá obraz, jestliže stav asociovaný na pozici $(l_1(p), l_2(p))$ odpovídá konečnému stavu.

Na obrázku 3.8 jsou zobrazeny první tři kroky teselačního automatu včetně inicilizačního kroku $t = 0$ (inicializační stav q_0 je reprezentován stavem I), v dalších krocích jsou pak zvýrazněny stavy, které v daném kroku budou zpracovávány.



Obrázek 3.8: Ukázka pohybu teselačního automatu

Kapitola 4

Determinizace teselačního automatu

Tato kapitola seznamuje s problémy determinizace dvoudimensionálního teselačního konečného automatu, rozebírá tyto problémy a následně nabízí řešení v podobě dvou algoritmů. První algoritmus generuje nedostupné stavy a transitivní funkce, druhý algoritmus nikoliv.

4.1 Rozbor nedeterminismu

Transitivní funkce má u nedeterministického teselačního automatu tvar

$$\delta : Q \times Q \times \Sigma \to 2^Q \text{ [5]}.$$

zatímco u deterministického teselačního automatu má tvar

$$\delta : Q \times Q \times \Sigma \to Q \text{ [5]}.$$

Nedeterminismus u teselačních automatů vzniká, pokud se pro stejnou vstupní část transitivní funkce vyskytuje více různých výstupních částí. Počet těchto výstupních částí pro některou transitivní funkci může být až 2^{Q}, kde 2^{Q} značí potenční množinu obsahující všechny podmnožiny množiny Q. Toto chování je pro jednoznačnost automatu nepřijatelné.

Příklad nedeterminismu u teselačních automatů je prezentován na příkladu níže.

Příklad 18. Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je nedeterminizovaný konečný dvoudimensionální teselační automat definovaný

$$\begin{gathered} Q = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}, \\ \Sigma = \{a, b\}, \\ q_0 = \{0\}, \\ F = \{5\}. \end{gathered}$$

s množinou transitivních funkcí δ obsahující transitivní funkce

$$\begin{gathered} \delta(0 \times 0 \times a) \to 1 \quad [1], \\ \delta(0 \times 0 \times a) \to 2 \quad [2]. \end{gathered}$$

pak je teselační automat nedeterministický, jelikož pro vstupní část $\delta(0 \times 0 \times a)$ existují dvě výstupní části definované stavy 1 a 2.

Při výskytu více výstupních částí pro zpracovávanou vstupní část transitivní funkce je třeba tyto výstupní části slučovat stejně jako u jednorozměrného konečného automatu.

Buňka teselačního automatu smí obsahovat až n sloučených či nesloučených stavů, kde n značí počet vstupních symbolů v množině vstupních symbolů Σ daného automatu. Počet těchto stavů v buňce závisí na tom, zda-li existují pro tyto symboly a zpracovávané stavy vyhovující pravidla.

Ukázka je demonstrována níže.

Příklad 19. Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je konečný nedeterminizovaný dvoudimensionální teselační automat z příkladu 18. Kromě transitivních funkcí v množině transitivních funkcí δ z tohoto příkladu obsahuje tato množina navíc ještě tyto transitivní funkce

$$\begin{aligned}
&\delta(1 \times 0 \times a) \to 3 \quad [3],\\
&\delta(1 \times 0 \times b) \to 4 \quad [4],\\
&\delta(0 \times 1 \times a) \to 5 \quad [5],\\
&\delta(0 \times 1 \times b) \to 6 \quad [6],\\
&\delta(0 \times 1 \times b) \to 7 \quad [7],\\
&\delta(0 \times 2 \times a) \to 8 \quad [8],\\
&\delta(0 \times 2 \times b) \to 9 \quad [9].
\end{aligned}$$

pak při zpracování první buňky vzniká při prvním (inicializačním) kroku transitivní funkce

$$\delta(0 \times 0 \times a) \to 1\_2 \quad [1, 2],$$

jelikož neexistuje žádná vyhovující inicializační transitivní funkce pro symbol b, bude v prvním inicializačním kroku vytvořena pouze tato jedna transitivní funkce. Za transitivní funkcí je uvedeno z jakých transitivních funkcí z původního nedeterministického teselačního automatu vznikla.

Zpracovávaná buňka (i, j) uchovává tento výstupní stav pro symbol a. Pokud by existovalo vhodné inicializační pravidlo pro symbol b, pak by bylo nutné v buňce uchovávat kromě stavu pro symbol a i stav pro symbol b.

Při druhém průchodu je zpracovávána buňka (i, j), kde

$$\begin{aligned}
&\text{buňka } (i-1, j) \text{ obsahuje stav } 0,\\
&\text{buňka } (i, j-1) \text{ obsahuje stav } 1_2.
\end{aligned}$$

Pro buňku (i, j) je nutné pro všechny symboly projít všechny kombinace kartézského součinu stavů v buňkách $(i-1, j)$ a $(i, j-1)$. Jedná se tedy o kombinace

$$\begin{aligned}
&0 \times 1,\\
&0 \times 2.
\end{aligned}$$

Všechny úplné vstupní části transitivních funkcí pro zpracovávanou buňku (i, j) v nedeterminizovaném automatu jsou

$$\begin{aligned}
&\delta(0 \times 1 \times a) \quad [10],\\
&\delta(0 \times 2 \times a) \quad [11],\\
&\delta(0 \times 1 \times b) \quad [12],\\
&\delta(0 \times 2 \times b) \quad [13].
\end{aligned}$$

výstupní části prvních dvou vstupních částí [10] a [11] z původního nedeterminizovaného automatu budou následně v determinizovaném automatu tvořit jeden výstupní stav. Pokud

v původním nedeterministickém automatu existují obě transitivní funkce s danou vstupní částí [10] a [11], pak dochází ke sloučení výstupních částí těchto funkcí a výsledný stav bude uložen v buňce (i, j) jako stav pro symbol a.

Obdobně je nutné proces opakovat i pro vstupní části [12] a [13] a pokud budou i tyto vstupní části v původním nedeterministickém automatu existovat, pak se jejich výstupní stav (sloučený či nesloučený podle počtu výskytů transitivních funkcí) uloží do téže buňky jako stav pro symbol b. Jak je prezentováno dříve, buňka tedy může uchovávat více stavů pro různé symboly.

Vstupní části transitivních funkcí [10] až [13] se v množině transitivních funkcí δ v původním nedeterminizovaném automatu vyskytují všechny. Dochází ke slučování všech výstupních stavů pro daný symbol a do determinizovaného automatu se přidávají transitivní funkce

$$\begin{aligned} &\delta(0 \times 1\_2 \times a) \to 4\_7 \quad [4, 7], \\ &\delta(0 \times 1\_2 \times b) \to 5\_6\_8 \quad [5, 6, 8]. \end{aligned}$$

Buňka (i, j) uchovává pro jednotlivé symboly tyto stavy

$$\begin{aligned} &a : 4\_7, \\ &b : 5\_6\_8. \end{aligned}$$

Stejným způsobem se provádí zpracování buněk i dále pro celý teselační automat.

Pro druhou buňku zpracovávanou v tomto kroku (i, j), kde

$$\begin{gathered} \text{buňka } (i-1, j) \text{ obsahuje stav } 1\_2, \\ \text{buňka } (i, j-1) \text{ obsahuje stav } 0, \end{gathered}$$

vznikají do determinizovaného teselačního automatu transitivní funkce

$$\begin{aligned} &\delta(1\_2 \times 0 \times a) \to 3 \quad [3], \\ &\delta(1\_2 \times 0 \times b) \to 4 \quad [4]. \end{aligned}$$

V následujícím kroku automatu by pro zpracovávanou buňku (i, j), kde

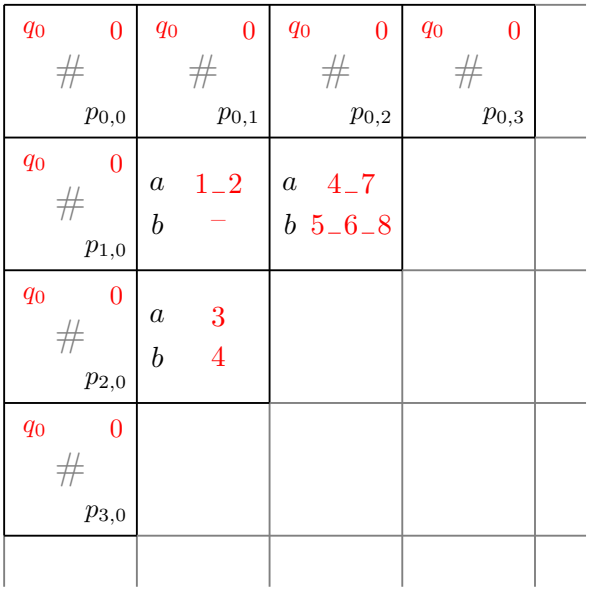
$$\begin{gathered} \text{buňka } (i-1, j) \text{ obsahuje stav } 0, \\ \text{buňka } (i, j-1) \text{ obsahuje stavy } 4\_7 \text{ pro symbol } a, \\ \text{buňka } (i, j-1) \text{ obsahuje stavy } 5\_6\_8 \text{ pro symbol } b, \end{gathered}$$

byly všechny úplné vstupní části transitivních funkcí v původním nedeterminizovaném teselačním automatu

$$\begin{aligned} &\delta(0 \times 4 \times a), \quad \delta(0 \times 4 \times b), \\ &\delta(0 \times 7 \times a), \quad \delta(0 \times 7 \times b), \\ &\delta(0 \times 5 \times a), \quad \delta(0 \times 5 \times b), \\ &\delta(0 \times 6 \times a), \quad \delta(0 \times 6 \times b), \\ &\delta(0 \times 8 \times a), \quad \delta(0 \times 8 \times b). \end{aligned}$$

celý postup popisovaný v tomto příkladu je znázorněn na obrázku 4.1.

Pokud je při determinizaci vytvořena nová transitivní funkce a je jeden z výstupních stavů shodný s některým stavem z původní množiny koncových stavů, pak je tento výstupní stav nově vytvořené transitivní funkce přidán do koncových stavů determinizovaného teselačního automatu.



Obrázek 4.1: Postup při odstraňování nedeterminismu

Příklad 20. Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je konečný determinizovaný dvoudimensionální teselační automat z příkladu 19. Při determinizaci byla vytvořena transitivní funkce

$$\delta(0 \times 1\_2 \times b) \to 5\_6\_8.$$

Původní nedeterministický automat má definovanou množinu koncových stavů

$$F = \{5\}.$$

Nový výstupní stav transitivní funkce (a tedy i nový stav determinizovaného automatu) má jeden ze stavů shodný s některým ze stavů z původní množiny koncových stavů nedeterministického automatu a proto bude stav 5_6_8 označen jako koncový stav determinizovaného automatu

4.2 Algoritmy determinizace

Následuje stručné vysvětlení syntaktických konstrukcí a principů jednotlivých algoritmů determinizace 6 a 7. Algoritmus 6 generuje transitivní funkce a stavy, které nebudou nikdy použity a z důvodu redundantní informace není vhodný. Pro implementaci v přiložené aplikaci, která je popsána v kapitole 5, je využit algoritmus 7, který nedostupné stavy a transitivní funkce negeneruje.

4.2.1 Algoritmus s nedostupnými stavy

Tento typ algoritmu je prezentován algoritmem 6. Příklad vstupu a výstupu po provedení procesu determinizace pomocí toho algoritmu je prezentován v příkladu 21.

Před zahájením algoritmu je do množiny stavů ke zpracování Q_{proc} vložen inicializační stav. Hlavní smyčka je se provede vzhledem ke své konstrukci vždy minimálně jednou pro inicializační typ transitivní funkce. Ve vnitřní logice algoritmu se přidávají stavy ke zpracování.

Pokud je množina stavů ke zpracování prázdná, pak algoritmus končí.

Z množiny stavů ke zpracování Q_{proc} se vyjme stav a v proměnné $Q_{current}$ přichází dále ke zpracování. Tento stav se vloží do množiny determinizovaných (výsledných) stavů Q_d.

Jelikož jsou transitivní funkce ve tvaru

$$\delta : (Q_1 \times Q_2 \times a) \rightarrow Q_3,$$

pak může být zpracovávaný stav buď na pozici Q_1 nebo Q_2. Do proměnných Q_{top} a Q_{left} se tedy uloží vyhovující dvojice se zpracovávaným stavem a symbolem v konstrukci pro každý symbol z množiny vstupních symbolů Σ. V následujících konstrukcích se prochází proměnné Q_{top} a Q_{left} a vyhledávají se výstupní části transitivních funkcí.

Dále se vytváří transitivní funkce s daným výstupním stavem. Pokud nalezený výstupní stav ještě nebyl zpracovaný (není obsažen v množině stavů Q_d), pak se vloží do množiny stavů ke zpracování Q_{proc}. Jestliže je v tomto stavu (může dojít ke sloučení stavů, výstupních stavů může být více) obsažen některý s původních koncových stavů nedeterminizovaného automatu, pak je tento výstupní stav vložen do koncových stavů determinizovaného automatu.

Algoritmus 6 determinizace teselačního automatu s nedostupnými stavy a funkcemi

Vstup: $M = (Q, \Sigma, \delta, q_0, F)$
Výstup: $M_d = (Q_d, \Sigma, \delta_d, q_0, F_d)$ - s nedostupnými stavy

```
1: repeat
2:     Q_proc := UNIQUE(Q_proc);
3:     Q_current := Q_proc;
4:     Q_proc := Q_proc − Q_current;
5:     Q_d := Q_d ∪ Q_current;
6:     for each a ∈ Σ do begin
7:         Q_top := {Q_1 : Q_2 ∈ Q_current, δ(Q_1 × Q_2 × a) → Q_3};
8:         Q_left := {Q_2 : Q_1 ∈ Q_current, δ(Q_1 × Q_2 × a) → Q_3};
9:         for each top ∈ Q_top do begin
10:            Q_out := {Q_3 : Q_1 ∈ top, Q_2 ∈ Q_current, δ(Q_1 × Q_2 × a) → Q_3};
11:            δ_d := δ_d ∪ δ(top × Q_current × a) → Q_out;
12:            if Q_out ≠ Q_d then Q_proc := Q_proc ∪ {Q_out};
13:            if Q_out ∩ F ≠ ∅ then F_d := F_d ∪ {Q_out}
14:        end for
15:        for each left ∈ Q_left do begin
16:            Q_out := {Q_3 : Q_1 ∈ Q_current, Q_2 ∈ left, δ(Q_1 × Q_2 × a) → Q_3};
17:            δ_d := δ_d ∪ δ(Q_current × left × a) → Q_out;
18:            if Q_out ≠ Q_d then Q_proc := Q_proc ∪ {Q_out};
19:            if Q_out ∩ F ≠ ∅ then F_d := F_d ∪ {Q_out}
20:        end for
21:    end for
22: until Q_proc = ∅;
23: δ_d := UNIQUE(δ_d);
24: F_d := UNIQUE(F_d).
```

4.2.2 Algoritmus pouze s dostupnými stavy a funkcemi

Tento typ algoritmu je prezentován algoritmem 7. Příklad vstupu a výstupu po provedení procesu determinizace pomocí tohoto algoritmu je prezentován v příkladu 21.

Hlavní smyčka algoritmu se provede vždy minimálně jednou pro inicializační stav, který musí být vždy součástí automatu pro spuštění procesu determinizace. Množina stavů $\mathrm{Q}_{\mathrm{proc}}$ reprezentuje aktuálně zpracovávanou diagonálu, respektive aktuální zpracovávaný krok teselačního automatu.

Před zpracováním každé diagonály se stavy z diagonály $\mathrm{Q}_{\mathrm{proc}}$ vkládají do množiny determinizovaných stavů Q_{d}. Buňky v diagonále se zpracovávají po dvojicích, přičemž před zpracováním každé diagonály jsou na začátek a konec množiny vloženy buňky obsahující inicializační stav, aby bylo dodrženo přirozené chování postupu automatu. Jednotlivé zpracovávané dvojice se mapují do proměnné $\mathrm{Q}_{\mathrm{map}}$ z důvodu testování konečné podmínky algoritmu.

Při procházení dvojic buňek diagonály $(i-1, j)$ a $(i, j-1)$ se vytváří kartézský součin všech stavů těchto buňek. Tyto buňky mohou obsahovat sloučené stavy a zároveň více stavů pro různé symboly. Následně se dvojice tohoto kartézského součinu prochází pro každý symbol $a \in \Sigma$. Pokud je nalezena vyhovující transitivní funkce (vstupní část tvoří právě zpracovávané stavy kartézského součinu a právě zpracovávaný vstupní symbol) v původním nedeterminizovaném teselačním automatu, pak se vytvoří nová transitivní funkce do determinizovaného teselačního automatu a výstupní stav (stav pro buňku (i, j)) pro daný symbol se přidá do množiny stavů $\mathrm{Q}_{\mathrm{step}}$. Množina $\mathrm{Q}_{\mathrm{step}}$ reprezentuje buňku automatu.

Pokud se jeden z výstupních stavů (může kvůli slučování stavů obsahovat těchto stavů více) shoduje s některým z původních koncových stavů, pak je přidán celý tento výstupní stav do koncových stavů determinizovaného automatu.

Po dokončení procházení kartézského součinu zpracovávané dvojice je množina $\mathrm{Q}_{\mathrm{step}}$ reprezentující buňku vložena do proměnné $\mathrm{Q}_{\mathrm{next}}$, která reprezentuje následující diagonálu, respektive následující krok teselačního automatu. Jakmile již nelze v aktuálním kroku zpracovat žádnou další dvojici, pak diagonála $\mathrm{Q}_{\mathrm{next}}$, která bude zpracovávána v příštím kroku označena jako aktuálně zpracovávaná a algoritmus pokračuje v následujícím průchodu se zpracováváním této nové diagonály.

Jednotlivé dvojice se mapují a následně se před zpracováváním další diagonály kontroluje toto mapování. Jestliže všechny dvojice v diagonále, která má být zpracovávána v příštím kroku, jsou již namapované (testuje se s přidáním dočasných buněk obsahující inicializační stavy na začátek a konec diagonály), pak nemá význam provádět zpracování této diagonály, jelikož by nevznikly žádné nové transitivní funkce ani stavy a algoritmus lze tedy ukončit ještě v tomto průchodu algoritmu.

Příklad 21. Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je nedeterminizovaný konečný teselační automat definovaný

$$\begin{aligned} Q &= \{0, 1, 2, 3\}, \\ \Sigma &= \{a\}, \\ q_0 &= \{0\}, \\ F &= \{1\}, \end{aligned}$$

s množinou transitivních funkcí δ obsahující transitivní funkce

$$\begin{array}{lll}
\delta(0 \times 0 \times a) \to 1, & \delta(2 \times 3 \times a) \to 1, & \delta(0 \times 1 \times a) \to 2, \\
\delta(0 \times 2 \times a) \to 2, & \delta(2 \times 1 \times a) \to 2, & \delta(2 \times 2 \times a) \to 2, \\
\delta(0 \times 1 \times a) \to 3, & \delta(1 \times 0 \times a) \to 3, & \delta(3 \times 0 \times a) \to 3, \\
\delta(1 \times 3 \times a) \to 3, & \delta(3 \times 3 \times a) \to 3. &
\end{array}$$

pak po procesu determinizace za použití algoritmu 6 vzniká determinizovaný teselační automat $M_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$ definovaný

$$\begin{aligned}
Q_1 &= \{0,\, 1,\, 3,\, 2,\, 23,\, 12,\, 13\}, \\
\Sigma &= \{a\}, \\
q_0 &= \{0\}, \\
F_1 &= \{1,\, 12,\, 13\},
\end{aligned}$$

s množinou transitivních funkcí δ_1 obsahující transitivní funkce

$$\begin{array}{lll}
\delta(0 \times 0 \times a) \to 1, & \delta(1 \times 0 \times a) \to 3, & \delta(3 \times 0 \times a) \to 3, \\
\delta(0 \times 1 \times a) \to 23, & \delta(0 \times 2 \times a) \to 2, & \delta(2 \times 1 \times a) \to 2, \\
\delta(1 \times 3 \times a) \to 3, & \delta(2 \times 3 \times a) \to 1, & \delta(3 \times 3 \times a) \to 3, \\
\delta(2 \times 2 \times a) \to 2, & \delta(2 \times 23 \times a) \to 12, & \delta(0 \times 23 \times a) \to 2, \\
\delta(1 \times 23 \times a) \to 3, & \delta(3 \times 23 \times a) \to 3, & \delta(23 \times 3 \times a) \to 13, \\
\delta(23 \times 1 \times a) \to 2, & \delta(23 \times 2 \times a) \to 2, & \delta(23 \times 0 \times a) \to 3, \\
\delta(0 \times 12 \times a) \to 23, & \delta(2 \times 12 \times a) \to 2, & \delta(12 \times 3 \times a) \to 13, \\
\delta(12 \times 1 \times a) \to 2, & \delta(12 \times 2 \times a) \to 2, & \delta(12 \times 0 \times a) \to 3, \\
\delta(2 \times 13 \times a) \to 12, & \delta(0 \times 13 \times a) \to 23, & \delta(1 \times 13 \times a) \to 3, \\
\delta(3 \times 13 \times a) \to 3, & \delta(13 \times 0 \times a) \to 3, & \delta(13 \times 3 \times a) \to 3.
\end{array}$$

Zatímco po provedení procesu determinizace pomocí algoritmu 7 vzniká determinizovaný teselační automat $M_2 = (Q_2, \Sigma, \delta_2, q_0, F_2)$ definovaný

$$\begin{aligned}
Q_2 &= \{0,\, 1,\, 23,\, 3,\, 2,\, 13,\, 12\}, \\
\Sigma &= \{a\}, \\
q_0 &= \{0\}, \\
F_2 &= \{1,\, 13,\, 12\},
\end{aligned}$$

s množinou transitivních funkcí δ_2 obsahující transitivní funkce

$$\begin{array}{lll}
\delta(0 \times 0 \times a) \to 1, & \delta(0 \times 1 \times a) \to 23, & \delta(1 \times 0 \times a) \to 3, \\
\delta(0 \times 23 \times a) \to 2, & \delta(23 \times 3 \times a) \to 13, & \delta(3 \times 0 \times a) \to 3, \\
\delta(0 \times 2 \times a) \to 2, & \delta(2 \times 13 \times a) \to 12, & \delta(13 \times 3 \times a) \to 3, \\
\delta(2 \times 12 \times a) \to 2, & \delta(12 \times 3 \times a) \to 13, & \delta(3 \times 3 \times a) \to 3, \\
\delta(2 \times 2 \times a) \to 2. &&
\end{array}$$

Je zřejmé, že první algoritmus 6 vygeneroval mnohem více transitivních funkcí a některé z nich nebudou vzhledem k postupu automatu nikdy použity. Tyto funkce jsou tedy nepotřebné a množství informací vygenerované prvním algoritmem je redundantní. Při nevhodně zvoleném nedeterminizovaném teselačním automatu může dojít i k vytváření nedostupných stavů, to ale není tento případ.

Jelikož druhý algoritmus 7 kopíruje chování automatu a jeho průchody dvoudimensionálním řetězcem, bude vždy generovat pouze dostupné transitivní funkce a stavy potřebné k fungování automatu.

Algoritmus 7 determinizace teselačního automatu s dostupnými stavy a funkcemi

Vstup: $M = (Q, \Sigma, \delta, q_0, F)$
Výstup: $M_d = (Q_d, \Sigma, \delta_d, q_0, F_d)$ - s dostupnými stavy

1: **repeat**
2: $Q_d := Q_d \cup Q_{proc}$;
3: $Q_{proc} := I \cup Q_{proc} \cup I$;
4: **repeat**
5: $Q_{top} \in Q_{proc}$
6: $Q_{proc} := Q_{proc} - \{Q_{top}\}$
7: $Q_{left} := Q_{proc}$
8: $Q_{map} := \text{MAP}(Q_{top}, Q_{left})$;
9: **for each** top $\in Q_{top}$ **do begin**
10: **for each** left $\in Q_{left}$ **do begin**
11: **for each** a $\in \Sigma$ **do begin**
12: $Q_{out} := \{Q_3 : Q_1 \in \text{top}, Q_2 \in \text{left}, \delta_d(Q_1 \times Q_2 \times a) \to Q_3 \in \delta\}$;
13: **if** $Q_{out} \neq \varnothing$ **then**
14: $\delta_d := \delta_d \cup \delta(\text{top} \times \text{left} \times a) \to Q_{out}$
15: $Q_{step} := Q_{step} \cup Q_{out}$;
16: **end if**
17: **if** $Q_{out} \cap F$ **then** $F_d := F_d \cup Q_{out}$;
18: **end for**
19: **end for**
20: **end for**
21: $Q_{next} := Q_{next} \cup Q_{step}$;
22: $Q_{step} = \varnothing$;
23: **until** $\text{COUNT}(Q_{proc}) > 1$;
24: $Q_{proc} := Q_{next}$;
25: $Q_{next} := \varnothing$;
26: **until** $\text{MAPCHECK}(Q_{map}, Q_{proc})$
27: $Q_d := \text{UNIQUE}(Q_d)$; $\delta_d := \text{UNIQUE}(\delta_d)$; $F_d := \text{UNIQUE}(F_d)$.

Kapitola 5

Aplikace

Následující kapitola se zabývá popisem výsledné aplikace na determinizaci konečného teselačního automatu. Bude představeno *uživatelské rozhraní*, zdůvodněn výběr programovacího jazyka a popsán způsob *implementace*.

5.1 Uživatelské rozhraní

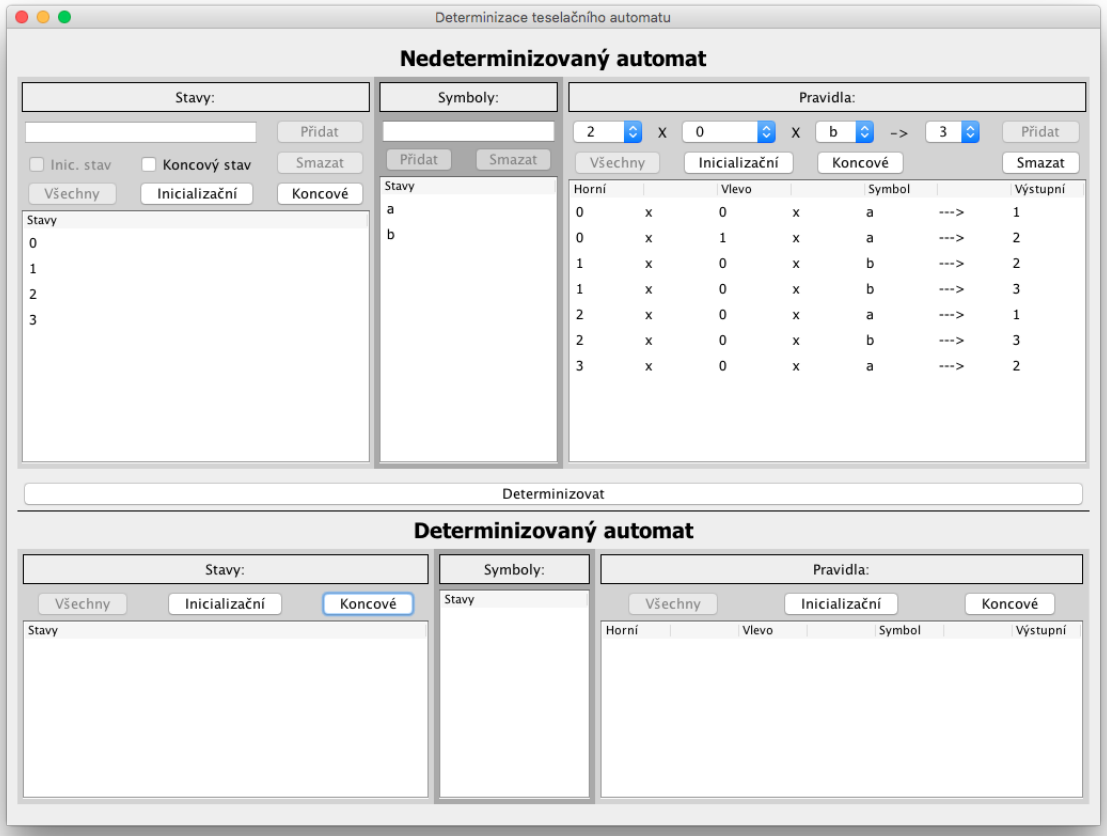
Základem programu je jednoduché a funkční uživatelské rozhraní. Pro celý proces determinizace je zapotřebí nízký počet funkcí, které aplikace nabízí v uživatelském rozhraní s elegantním, jednoduchým a přehledným *designem*.

Při spuštění aplikace se zobrazí vodorovně rozdělené okno. Horní polovina vyobrazuje aktuální hodnoty vstupního nedeterminizovaného automatu, dolní následně aktuální hodnoty výstupního determinizovaného automatu. Mezi těmito polovinami se nachází tlačítko k zahájení procesu determinizace. Vstupní i výstupní automat zobrazuje tři základní komponenty teselačního automatu — stavy, symboly a transitivní funkce. Všechny tyto komponenty a jejich vyobrazení budou prezentovány dále. Pokud je vstupní nedeterminizovaný automat prázdný, pak nelze spustit proces determinizace. Toto chování je indikováno neaktivním tlačítkem **Determinizovat** uprostřed aplikace. Aby bylo možné determinizaci spustit, pak musí vstupní nedeterminizovaný automat obsahovat nejméně jeden stav v množině stavů, který je zároveň označen jako inicializační. Na obrázku 5.1 je pro úvodní představu vyobrazeno uživatelské rozhraní aplikace s ukázkovými daty automatu.

5.1.1 Stavy

Komponenta stavů zahrnuje textové pole pro zadání názvu stavu, několik doplňujících možností pro specifikaci stavu či filtrování stavů a zásobník stavů pro rychlou orientaci v již vložených stavech automatu. Všechny tyto prvky rozhraní jsou pro větší přehlednost vyobrazeny na obrázku 5.3.

Přidání stavu se provádí pomocí stisku tlačítka **Přidat**. Pokud je vstupní pole prázdné, pak nelze stav přidat, jelikož není pojmenovaný. Pokud je zadán název stavu, který již v automatu existuje, pak rovněž nebude možné tento stav přidat. Tyto dvě možnosti, kdy nelze stav přidat jsou simulovány neaktivním tlačítkem **Přidat**, které nelze v těchto situacích stisknout. Jestliže bude zadán název stavu, který se v automatu zatím nevyskytuje, pak se tlačítko **Přidat** automaticky změní na aktivní a uživateli tímto způsobem dává zpětnou vazbu, že lze tento stav do automatu přidat. Reakce tlačítka **Přidat** na různé vstupy v textovém poli je uvedena na obrázku 5.2.



Obrázek 5.1: Ukázka uživatelského rozhraní aplikace

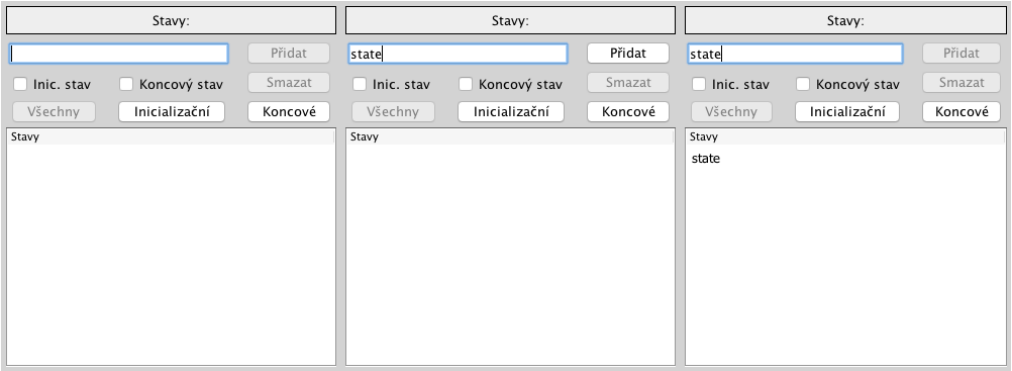
Před samotným přidáním stavu lze zvolit ze dvou možností, které udávají vlastnosti daného stavu. První možností je zaškrtnutí pole **Inic. stav**, v tom případě bude stav označen jako inicializační. Pokud se už bude v automatu jeden inicializační stav vyskytovat, pak aplikace nedovolí uživateli další inicializační stav přidat a simuluje toto chování zneaktivněním této možnosti. Druhou možností je označit vkládaný stav jako koncový pomocí zaškrtnutí volby **Koncový stav**, potom bude stav přidán jako koncový. Na tuto možnost se omezení nevztahují a uživatel smí definovat libovolný počet koncových stavů.

Poněvadž může uživatel vložit chybný stav, lze tento stav smazat označením daného stavu v zásobníku stavů a stisknutím tlačítka **Smazat**. Pokud není v zásobníku stavů vybrán žádný stav, pak se tlačítko **Smazat** automaticky zneaktivní. Naopak, tlačítko **Smazat** se automaticky zaktivní jakmile bude označen některý ze stavů v zásobníku.

Následující možnosti již nepřidávají specifikace pro přidávaný stav, avšak nabízí možnost filtrování již vložených stavů. Zde jsou na výběr tři možnosti. Zobrazit **Všechny** stavy, **Inicializační** stav či **Koncové** stavy. Aktuální výběr indikuje zneaktivnění daného tlačítka.

Jednotlivé prvky komponenty stavů ilustrované na obrázku 5.3:

1. Textové pole pro vložení názvu stavu
2. Tlačítko **Přidat**
3. Tlačítko **Smazat**
4. Zaškrtávací box **Inicializační stav**
5. Zaškrtávací box **Koncový stav**
6. Filtr **Všechny** stavy
7. Filtr **Inicializační** stavy
8. Filtr **Koncové** stavy
9. Zásobník vložených stavů



Obrázek 5.2: Reakce mezi vstupem v textovém poli a tlačítkem **Přidat**

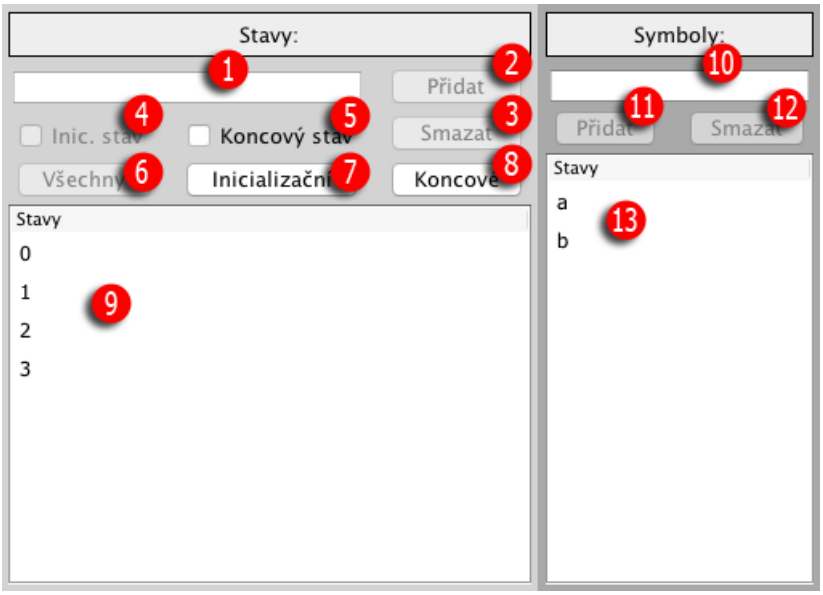
5.1.2 Vstupní symboly

Komponenta vstupních symbolů nabízí textové pole pro název symbolu, který musí být jednoznakový a tlačítka **Přidat** či **Smazat**. Funkčnost tlačítek **Přidat** a **Smazat** je shodná s funkčností u komponenty stavů, nemá tedy smysl ji hlouběji rozebírat.

Popis prvků komponenty vstupních symbolů je prezentován níže, tyto prvky jsou ilustrovány na obrázku 5.3.

Jednotlivé prvky komponenty vstupních symbolů:

10. Textové pole pro vložení názvu vstupního symbolu
11. Tlačítko **Přidat**
12. Tlačítko **Smazat**
13. Zásobník vložených symbolů



Obrázek 5.3: Komponenty stavů a symbolů a jejich popis

5.1.3 Transitivní funkce

Komponenta transitivních funkcí nabízí možnost přidávat jednotlivé transitivní funkce. Skládá se ze čtyř výběrových polí pro specifikaci transitivní funkce, tlačítek **Přidat** a **Smazat**, možností pro filtrování transitivní funkce a zásobníku transitivních funkcí.

Pro přidání transitivní funkce je podle předpisu transitivní funkce

$$\delta : Q \times Q \times \Sigma \to Q$$

nutné zadat dva vstupní stavy, vstupní symbol a výstupní stav, aby tato transitivní funkce mohla být pomocí tlačítka **Přidat** přidána. Tyto stavy a symbol musí být nejdříve přidané v komponentě stavů, respektive symbolů. Pokud ještě transitivní funkce v zásobníku transitivních funkcí neexistuje, pak ji lze do zásobníku přidat pomocí stisknutí tlačítka **Přidat**. Pokud již v zásobníku existuje, pak je tlačítko **Přidat** zneaktivněno a uživateli není umožněno jej stisknout.

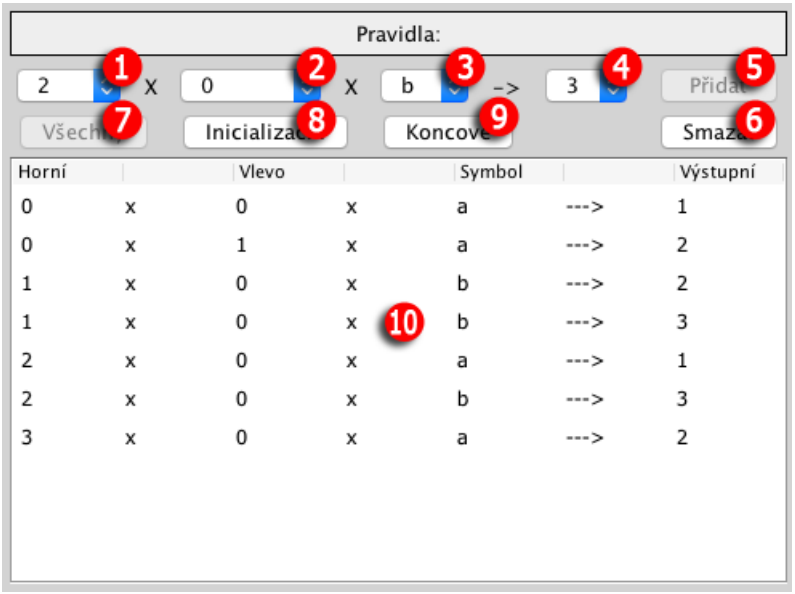
Pokud je označena některá z transitivních funkcí v zásobníku, pak ji lze smazat pomocí tlačítka **Smazat**. Pokud není v zásobníku označena žádná, pak je tlačítko zneaktivněno.

Popis všech prvků komponenty transitivních funkcí je popsán níže a tyto prvku jsou ilustrovány na obrázku 5.4.

Jednotlivé prvky komponenty transitivních funkcí:

1. Pole pro výběr vstupního stavu $(i-1, j)$
2. Pole pro výběr vstupního stavu $(i, j-1)$
3. Pole pro výběr vstupního symbolu
4. Pole pro výběr výstupního stavu (i, j)
5. Tlačítko **Přidat**

6. Tlačítko **Smazat**
7. Filtr **Všechny** transitivní funkce
8. Filtr **Inicializační** transitivní funkce. Jedná se o funkce, které v jednom ze stavů vstupní části obsahují inicializační stav
9. Filtr **Koncové** transitivní funkce. Jedná se o funkce, ve kterých je výstupní stav stavem koncovým
10. Zásobník vložených transitivních funkcí



Obrázek 5.4: Komponenta transitivních funkcí a její popis

5.2 Implementace

Aplikace je implementována v *Javě*. Java je objektovně orientovaný programovací jazyk. Jedná se o velmi rozšířený jazyk a v současné době také o nejpoužívanější programovací jazyk na světě. Největší výhodou Javy je bezesporu její *přenositelnost* a nezávislost na konkrétní *platformě*. Tento jazyk je také velice bezpečný co se týče z hlediska tvoření potenciálně nebezpečných syntaktických konstrukcí a přetypování. Jazyk byl dále zvolen kvůli elegantnímu a lehce čitelnému kódu. Syntaxe Javy je jednoduchá, jelikož součástí jazyka nejsou některé ze základních programovacích součástí jazyků *C/C++*, např. *preprocesor* či *ukazatele*.

Implementace probíhala ve dvou částech. První část definuje naprogramování vnitřní logiky reprezentováno hlavní třídou vnitřní logiky `Controller`. Tato část zahrnovala vytvoření optimálního rozhraní pro stavy, symboly a transitivní funkce a implementaci procesu determinizace podle algoritmu 7 z předcházející kapitoly. Druhou část definuje vytvoření grafického uživatelského rozhraní pro snazší komunikaci mezi vnitřní logikou aplikace a uživatelem. Tato část je reprezentována hlavní grafickou třídou `AppWindow`. Obě tyto části jsou uzavřeny v hlavní třídě programu zvané `Determinization`.

5.2.1 Implementace vnitřní logiky

Hlavní třídou vnitřní logiky je třída `Controller`, která zajišťuje hlavní kontrolu nad programem. Uchovává informace o vstupním nedeterminizovaném konečném automatu i výsledném výstupním konečném determinizovaném automatu, jednotlivé informace o právě zpracovávaném kroku teselačního automatu a informace o mapování dvojic. Zároveň je v této třídě implementována nejdůležitější funkce celého programu — samotný proces determinizace a jeho algoritmus.

Třída `Controller` využívá pro uchování informací o automatu třídu `Automaton`. Třída `Automaton` ukládá jednotlivé transitivní funkce do instance třídy `RuleStack` a jednotlivé symboly do instance třídy `SymbolStack`. Smyslem těchto tříd, `RuleStack` a `SymbolStack`, je pouze uschovat jednotlivé instance `Rule` pro transitivní funkci, respektive `Symbol` pro symbol, a nabízet sadu operací nad těmito instancemi, např. vrátit množinu těchto instancí či tyto instance seřadit. Třídy `RuleStack` a `SymbolStack` využívají dědičnosti generické třídy `ArrayList<>`.

Třída `Symbol` reprezentuje jeden vstupní symbol automatu, kdy využívá *konstruktoru* s jediným parametrem typu *char*, symboly tedy mohou být pouze jednoznakové. Třída `Rule` uchovává informace o transitivní funkci, která se skládá ze třech stavů, dvou vstupních a jednoho výstupního, a jednoho symbolu. Této *signatuře* odpovídá i jediný konstruktor této třídy. Třídy `Rule` i `Symbol` implementují generické rozhraní `Comparable` pro možnost využití seřazení pomocí statické metody *sort* třídy `Collections` pro seřazení množiny symbolů/transitivních funkcí.

Pro implementaci stavů automatu je zvoleno odlišné řešení. Celkem bylo implementováno pět tříd pro zpracování stavů a front těchto stavů. Všechny tyto třídy jsou přehledně zobrazeny v tabulce 5.1.

Jelikož může dojít při procesu determinizace ke slučování výstupních stavů, pak je nutné pracovat s instancí třídy, která je schopna uchovávat více než jeden stav. Třída vytvářející tyto instance je nazývána `StateSet`. Tato třída je výchozí třídou pro reprezentaci stavu v automatu. Jednotlivé stavy, které třída `StateSet` uchovává jsou reprezentovány třídou `State`. Třída `State` nabízí dva konstruktory. První konstruktor přebírá jeden parametr typu *String*, který reprezentuje název stavu a jedná se o stav, který nemá funkci inicializačního ani koncového stavu. Druhý konstruktor kromě parametru *String* pro deklarování názvu stavu nabízí možnost vytvořit inicializační či koncový stav pomocí dalších parametrů typu `Role`. Enumerace `Role` nabízí dvě hodnoty

`INIT` pro inicializační stav,
`FINITE` pro koncový stav.

Jedna buňka teselačního automatu může obsahovat při determinizaci více stavů z důvodu přítomnosti více symbolů v automatu. Z tohoto důvodu se v programu vyskytuje třída `StateQueue`. Třída `StateQueue` reprezentuje jednu buňku teselačního automatu a uchovává instance třídy `StateSet`, tedy jednotlivé stavy. Tato třída nabízí množinu operací nad uchovávanými instancemi stavů, např. řazení stavů, odstranění duplicit stavů či různé druhy odstraňování stavů z množiny.

Zpracovávaný krok teselačního automatu lze vykreslit jako diagonálu, kterou prezentuje třída `DiagonalQueue`. Třída `DiagonalQueue` uchovává instance třídy `StateQueue`, tedy jednotlivé buňky teselačního automatu a stavy v nich. Tato třída nabízí také sadu operací nad množinou uchovávaných instancí. Jednou z těchto operací je vložení inicializačního stavu na začátek i konec fronty tak, aby byla správně simulováno zpracování diagonály

teselačním automatem.

Poslední třídou pracující s množinou stavů je třída `MapQueue`. Tato třída slouží k mapování dvojic a testování podmínky pro ukončení algoritmu determinizace, jestliže již nelze zpracovat nové dvojice v příštím kroku automatu a tím vytvořit nové transitivní funkce či nové stavy.

Třída	**Význam třídy**
`State`	Jednotlivý stav
`StateSet`	Set reprezentující stav automatu
`StateQueue`	Buňka teselačního automatu
`DiagonalQueue`	Diagonála teselačního automatu
`MapQueue`	Mapování zpracovaných dvojic automatu

Tabulka 5.1: Shrnutí tříd pracujících se stavy či s množinou stavů

5.2.2 Implementace grafického rozhraní

Pro tvorbu grafického uživatelského rozhraní jsou použity Java knihovny *AWT* a *Swing*. K rozvržení jednotlivých částí hlavního okna programu je použit především *GridBagLayout*, který dovoluje absolutní kontrolu nad rozvržením. Sekundárně je pro jednodušší rozvržení použit *BorderLayout*. Pro zpracování událostí jsou použity třídy `ActionListener` pro tlačítka či `KeyListener` pro vstupní textové pole a následné přepsání jejich metod. V tabulce 5.2 jsou uvedeny základní třídy knihovny Swing pro tvorbu prvků grafického uživatelského rozhraní.

Jako hlavní třídu celého programu lze označit třídu `Determinization` dědící třídu `Canvas` pro vytvoření uživatelského rozhraní. V této třídě se vytváří instance hlavní logické třídy `Controller` pro vykonávání vnitřní logiky a instance třídy `AppWindow` dědící třídu `JFrame` definující hlavní okno aplikace. Třída `AppWindow` určuje specifické vlastnosti hlavního okna aplikace, zejména nastavení šířky a výšky okna.

Třída `MainPanel` rozděluje hlavní okno programu na tři části. Horní a dolní část okna aplikace je vyhrazena pro vstupní, respektive výstupní automat definovaný grafickou třídou `AutomatonPanel`. Prostor v prostřední části je vytvořen pro tlačítko zahajující proces determinizace, „Determinizovat“.

Třída `AutomatonPanel` vyplňuje horní část svého přiděleného prostoru pomocí specifického nadpisu automatu („nedeterministický“, „deterministický“) a zbylou část pomocí komponent automatu (stavů, symbolů a transitivních funkcí). Tyto komponenty mají několik společných prvků, proto bloky `StateBlock`, `SymbolBlock`, `RuleBlock` reprezentující jednotlivé komponenty, dědí ze stejné abstraktní třídy `AutomataPart`.

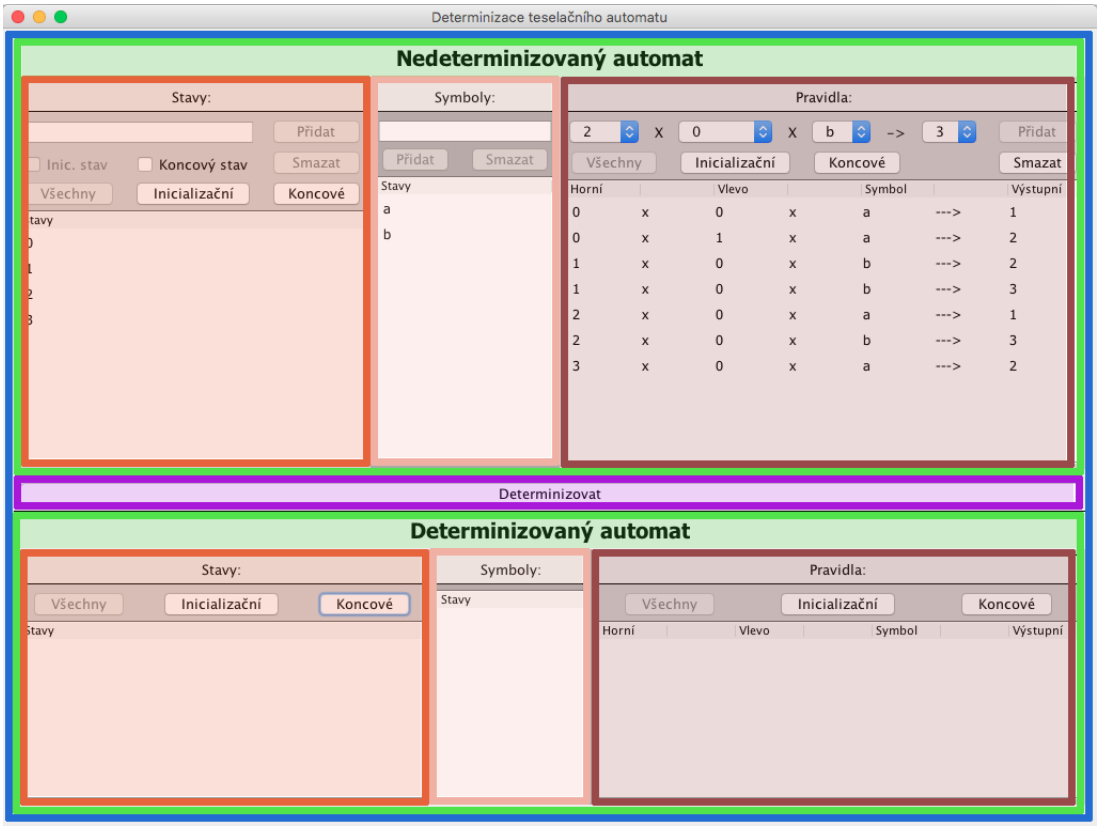
Rozvržení tříd v hlavním okně aplikace je pro přehlednost ilustrováno na obrázku 5.5. Odpovídající legenda k obrázku je v tabulce 5.3.

Prvek rozhraní	Odpovídající grafická třída
tlačítko	JButton
textové pole	JTextField
zaškrtávající pole	JCheckBox
pole pro výběr	JComboBox
tabulka	JTable

Tabulka 5.2: Prvky grafického uživatelského rozhraní knihovny Swing

Třída grafického rozhraní	Znázorňující barva
`MainPanel`	
`AutomatonPanel`	
`StateBlock`	
`SymbolBlock`	
`RuleBlock`	
tlačítko „Determinizovat“	

Tabulka 5.3: Legenda k rozvržení tříd



Obrázek 5.5: Rozvržení hlavního okna aplikace

Kapitola 6

Závěr

V této práci byl poskytnut základní pohled na jednorozměrné i dvourozměrné formální jazyky a rozdíly mezi nimi. Byly definovány jednorozměrné a dvourozměrné konečné automaty a představeny jejich základní typy. V teorii jednorozměrných automatů byl vysvětlen problém nedeterminismu a byly popsány již známé možnosti řešení.

Jádrem této práce bylo rozebrání možného nedeterminismu u teselačního automatu, který patří do skupiny celulárních automatů. Po důkladném prozkoumání tohoto problému byly navrženy dva algoritmy. Smyslem prvního algoritmu je vytvářet všechny podmnožiny stavů a k nim pak doplňovat transitivní funkce. Vznikají tedy zbytečné redundantní informace, které automatem nikdy nebudou využity. Druhý algoritmus kopíruje přirozený postup automatu a podle tohoto postupu vytváří nové stavy a transitivní funkce. V demonstrovaném příkladu jsou následně na jednom nedeterminizovaném teselačním automatu využity oba algoritmy a jsou popsány jejich rozdílné výsledky.

Algoritmus, který kopíruje přirozený postup automatu, byl použit v implementaci přiložené aplikace. Aplikace slouží k provedení procesu determinizace nad vloženým nedeterministickým teselačním automatem. Jako implementační jazyk byla použita Java JDK 8. Aplikace nabízí přehledné a pro obsluhu jednoduché uživatelské rozhraní. Byl popsán postup implementace této aplikace od vnitřní logiky až po implementaci grafického rozhraní.

Na tuto práci lze navázat vytvořením algoritmů pro proces determinizace jiného z dvoudimensionálních automatů. Příkladem může být čtyřcestný dvoudimensionální automat, který je v této práci představen. Dvoudimensionální automaty nabízí různé modifikace pro vyšší výkonnost a existuje jich nepřeberné množství. Pro tyto algoritmy lze následně vytvořit aplikaci s grafickým uživatelským rozhraním pro snažší komunikaci mezi vnitřní logikou aplikace a uživatelem. Druhou možností, jak lze na tuto práci navázat, je pokračovat ve zpracování teselačního automatu a vytvořit algoritmy jeho minimalizace.

Literatura

[1] Hopcroft, J. E.; Motwani, R.; Ullman, J. D.: *Automata Theory, Languages, and Computation*. Pearson Education, Inc., 2007, ISBN 0-321-455-36-3.

[2] Koutný, J.: Dvourozměrné jazyky a digitální obrazy. FIT VUT v Brně, [Online; navštíveno 13.4.2016].
URL http://www.fit.vutbr.cz/study/courses/TJD/public/0809TJD-Koutny.pdf

[3] Meduna, A.: *Automata and Languages*. Springer, 2000, ISBN 1-85-233-074-0.

[4] Průša, D.: *Two-dimensional Languages*. Dizertační práce, Charles University, 2004.
URL http://cmp.felk.cvut.cz/ftp/articles/prusa/Two-dimensional%20Languages.pdf

[5] Rozenberg, G.; Salomaa, A.: *Handbook of Formal Languages*. Springer, 1997, ISBN 3-540-60649-1.

[6] Vaníček, J.; Papík, M.; Pergl, R.; aj.: *Teoretické základy informatiky*. Kernberg Pusblishing s.r.o., 2007, ISBN 978-80-903962-4-1.

Přílohy

Seznam příloh

A Obsah DVD **48**

Příloha A

Obsah DVD

- `readme.txt` — soubor s popisem obsahu DVD a návodem k aplikaci
- `build.xml` — soubor využívaný programem `ant` k sestavení a spuštění aplikace
- `\src` — adresář se zdrojovými kódy aplikace
- `\bin` — adresář se spustitelnou aplikací
- `\thesis_pdf` — adresář s písemnou zprávou ve formátu Portable Document Format
- `\thesis_src` — adresář se zdrojovými kódy písemné zprávy v LaTeX