VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMEDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

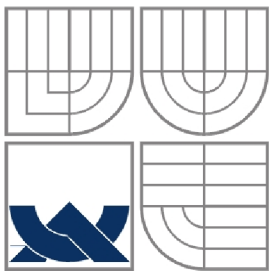
VIZUALIZACE PRÁCE KONEČNÝCH AUTOMATŮ, ZÁSOBNÍKOVÝCH AUTOMATŮ A TURINGOVA STROJE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

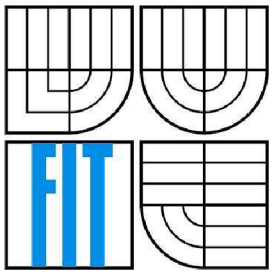
AUTOR PRÁCE
AUTHOR

ONDŘEJ SYROVÝ

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMEDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VIZUALIZACE PRÁCE KONEČNÝCH AUTOMATŮ, ZÁSOBNÍKOVÝCH AUTOMATŮ A TURINGOVA STROJE

VISUALIZATION OF FINITE AUTOMATA, PUSHDOWN AUTOMATA AND TURING MACHINES WORK

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ONDŘEJ SYROVÝ
AUTHOR

VEDOUCÍ PRÁCE — ING. JIŘÍ ZUZAŇÁK
SUPERVISOR

BRNO 2009

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2008/2009

Zadání bakalářské práce

Řešitel: **Syrový Ondřej**
Obor: Informační technologie
Téma: **Vizualizace práce konečných automatů, zásobníkových automatů a Turingova stroje**
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte základní problematiku teorie zabývající se konečnými automaty, zásobníkovými automaty a Turingovými stroji
2. Navrhněte rozhraní pro popis těchto systémů, včetně jejich vstupů a počátečních stavů
3. Navrhněte a implementujte virtuální stroj simulující tyto systémy s grafickou reprezentací aktuálního stavu a přechodů těchto strojů
4. Zhodnoťte dosažené výsledky a diskutujte možnost využití vytvořeného programu jako demonstrační pomůcky při výuce funkce těchto strojů

Literatura:

- po domluvě s vedoucím

Při obhajobě semestrální části projektu je požadováno:

- 1. - 2. bod zadání

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: **Zuzaňák Jiří, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2008
Datum odevzdání: 20. května 2009

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce se zabývá návrhem a implementací aplikace pro demonstraci činnosti konečných automatů, zásobníkových automatů a Turingova stroje. Teoretická část práce se zabývá teorií formálních jazyků, gramatik a automatů. Vytvořený program umožňuje načítání deterministických i nedeterministických variant automatů ze souboru, jejich grafickou reprezentaci pomocí stavového diagramu, krokování výpočtu a znázornění možných přechodů.

Abstract

This bachelor`s thesis is focusing on concept and development of computer application for demonstration of finite automata, pushdown automata and Turing machines work. Theoretic volume of this work deals with theories of formal languages and grammars and automata theory. Created program allows to load deterministic and nondeterministic automata variants from the text file, their graphic representation by state diagram and stepping their calculation process.

Klíčová slova

Konečný automat, zásobníkový automat, Turingův stroj, demonstrace, determinizmus, simulace, formální jazyk, gramatika, teorie automatů, wxWidgets

Keywords

Finite automata, pushdown automata, Turing machine, demonstration, determinism, simulation, formal language, grammar, automata theory, wxWidgets

Citace

Syrový Ondřej: Vizualizace práce konečných automatů, zásobníkových automatů a turingova stroje, bakalářská práce, Brno, FIT VUT v Brně, 2009

Vizualizace práce konečných automatů, zásobníkových automatů a turingova stroje

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením ing. Jiřího Zuzaňáka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Ondřej Syrový
16.5.2009

Poděkování

Rád bych poděkovat Ing. Jiřímu Zuzaňákovi za jeho odbornou pomoc, kterou mi poskytoval během vytváření této práce.

Obsah

1 Úvod

Teorie automatů je obsáhlá disciplína, která zahrnuje pro laika velké množství nových pojmů a na první pohled složitých definicí. Přitom jádro této disciplíny, tedy stavové automaty a princip jejich práce, se dá velice názorně demonstrovat pomocí vhodných vizuálních reprezentací (především pak stavových diagramů). Tím, že student pochopí, jakým způsobem automaty pracují, získá nadhled, který mu pomůže rychleji a snáz proniknout do obsáhlé problematiky, která kolem automatů existuje. A to je také cílem této práce. Vytvořit aplikaci pro vizualizaci práce konečných automatů (KA), zásobníkových automatů (ZA) a Turingových strojů (TS), která si klade za cíl pomoct studentům rychle se zorientovat při jejich prvním seznámení s teorií automatů. Tato aplikace by měla nalézt uplatnění především jako demonstrační pomůcka při výuce.

Počátky teorie automatů spadají do období druhého čtvrtletí dvacátého století, tedy ještě dříve než se začala rýsovat koncepce počítačů tak, jak je známe dnes. Tato disciplína ovlivnila spoustu vědních oborů (například matematiku, logiku, lingvistiku, biologii a další.) a značně se promítla do vývoje moderní informatiky.

V informatice se automaty nejčastěji využívají v oblasti programovacích jazyků, kde pracují jako akceptory jazyků při překladu. Co jsou to formální jazyky a jakými prostředky je můžeme specifikovat je uvedeno ve druhé kapitole. Ta je koncipována jako teoretický úvod do problematiky jazyků a gramatik a dále obsahuje Chomského hierarchii, podle které rozdělujeme jak gramatiky, tak i jazyky.

Ve třetí kapitole se nachází krátký přehled historie automatů a kritéria, podle kterých je možné automaty rozdělovat. Dále obsahuje tři podkapitoly, kde jsou blíže popsány jednotlivé typy automatů, jejichž vizualizace je předmětem této práce.

Na vlastní implementaci aplikace je zaměřena čtvrtá kapitola. Začíná specifikací požadavků a výběrem vývojového prostředí. Dále obsahuje návrh rozhraní pro popis automatů a návrh uživatelského rozhraní. V poslední podkapitole je pak popsáno, jakým způsobem byla řešena grafická reprezentace automatů.

V páté kapitole najdete popis procesu testování aplikace a její funkčnosti. V poslední kapitole pak závěrečné shrnutí a zhodnocení dosažených výsledků.

2 Jazyky a gramatiky

2.1 Formální jazyky

Pod pojmem jazyk si většinou představíme množinu slov a gramatik, které používají jednotlivé národnosti ke komunikaci a dorozumívání. Těmto jazykům se říká jazyky přirozené a jsou to například angličtina a čeština. Přirozené jazyky obsahují množinu slov, které se podle gramatiky daného jazyka slučují ve věty. U jazyků formálních věty ze slov netvoříme, tedy jednotlivé symboly můžeme chápat jako slova a slova pak jako věty. Neformálně můžeme říct, že pokud jakkoliv vymezíme množinu řetězců, získáme formální jazyk. Pro formální vyjádření si musíme definovat co je to abeceda, symbol a řetězec.
Základní definice:

Definice 2.1.2
Abeceda Σ je libovolná neprázdná konečná množina, jejíž prvky nazýváme symboly.

Definice 2.1.3
Pod pojmem řetězec w (někdy také „slovo") je pak chápána každá konečná posloupnost symbolů nad abecedou Σ. Prázdná posloupnost se nazývá prázdný řetězec a většinou se značí písmenem ε.

Nad řetězci můžeme provádět asociativní operaci zvanou zřetězení (neboli konkatenace). Jedná se o spojení dvou a více slov. Mějme například dva řetězce w a x. Zřetězení těchto slov se zapisuje jako w. x (nebo zkráceně wx). Pokud například w = 01 a x = 11, pak jejich zřetězením získáme řetězec 0111, tedy w.x = 0111.

Délka řetězce udává počet jeho symblů. Značí se symbolicky |x|. Je-li $x = a_1a_2\ldots a_n$, $a_i \in \Sigma$ pro i = 1,…n, pak |x| = n. Délka prázdného řetězce je nulová.

Řetězec u je podřetězcem řetězce v, jestliže existují řetězce x, y takové, že v = xuy. Pokud navíc x = ε, říkáme že řetězec u je předponou (prefixem) řetězce v a pokud y = ε, pak je řetězec u příponou (sufixem) řetězce v. Například pro řetězec `abcd` jsou to:

Podřetězce: `ε, a, b, c, d, ab, bc, cd, abc, bcd, abcd.`
Předpony: `ε, a, ab, abc, abcd.`
Přípony: `ε, d, cd, bcd, abcd.`

Množinu všech slov nad abecedou Σ značíme Σ^*, množinu všech neprázdných slov pak Σ^+. Nyní si již můžeme nadefinovat jazyk:

Definice 2.1.1
Nechť **Σ*** značí množinu všech řetězců nad abecedou **Σ**. Množinu L pro níž platí $\mathbf{L \subseteq \Sigma^*}$ nazýváme jazykem nad abecedou **Σ**.

Přitom prázdná množina Ø a množina obsahující pouze prázdný řetězec {ε}, jsou jazyky nad každou abecedou.

Nad jazyky můžeme provádět běžné množinové operace jako je průnik, sjednoceí a rozdíl, a stejně jako u řetězců operace zřetězení a iterace.

Jazyky můžeme rozdělit na konečné, tedy takové, které obsahují konečně mnoho řetězců, a jazyky nekonečné, které obsahují nekonečné množství řetězců. Pokud bychom chtěli specifikovat konečný jazyk, vystačili bychom si s výčtem slov. Ale spousta jazyků, s kterými potřebujeme pracovat, například většina programovacích jazyků, spadá mezi jazyky nekonečné. K tomu, abychom mohli obecně nekonečné jazyky specifikovat konečnými prostředky, slouží gramatiky (pro generování jazyka) a automaty (pro jeho rozpoznání).

2.2 Gramatiky

Definice 2.2.1

Gramatika G je čtveřice **G = (N,T,P,S)**, kde

- **N** je neprázdná konečná množina neterminálních symbolů (abeceda neterminálů)
- **T** je konečná množina terminálních symbolů (abeceda terminálů), kde $N \cap T = \emptyset$
- **P** je množina přepisovacích pravidel, která je konečnou množinou podmnožiny $P \subseteq (N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$
- **S** je speciální počáteční neterminál, $S \in N$

Nonterminály mají roli pomocných proměnných, které označující určité syntaktické celky (syntaktické kategorie).

Terminály jsou symboly, jejichž množina je identická s abecedou, nad kterou je definován jazyk. Proto se také často označuje jako abeceda symbolem Σ.

Slovník gramatiky je pojem označující výslednou množinu danou sjednocením množin terminálů a nonterminálů ($N \cup T$).

Gramatika je tedy soubor (přepisovacích) pravidel, podle kterých lze generovat jednotlivá slova daného jazyka. Pravidla obvykle zapisujeme ve tvaru $\alpha \rightarrow \beta$, kde levá strana (tedy α) musí obsahovat alespoň jeden neterminál.

Určitý jazyk lze popsat vícero ekvivalentními gramatikami, proto je vhodné gramatiky zkoumat nejen podle toho jaký jazyk popisují, ale také jakým způsobem to dělají. Nejznámější klasifikace gramatik je tzv. Chomského hierarchie, která rozděluje gramatiky do čtyř tříd, které generují různé podskupiny formálních jazyků.

Hlavním rozdílem mezi jednotlivými automaty je jejich výpočetní síla. Ta je dána právě typem jazyka, který dokáže rozpoznat. Jaké jsou třídy gramatik, jazyků a které automaty je dokážou identifikovat je popsáno v další podkapitole.

2.3 Chomského hierarchie

Chomského hierarchie byla vytvořena roku 1956 Noamem Chomskym. Rozděluje čtyři typy jazyků a klasifikuje čtyři typy gramatik podle tvaru přepisovacích pravidel: typ 0, typ 1(kontextová), typ 2(bezkontextová) a typ 3(regulární). Hierarchičnost pak spočívá v tom, že každý regulární jazyk je jazykem bezkontextovým, každý bezkontextový je poté jazykem kontextovým a každý kontextový jazyk je jazykem typu 0 (viz. Obr. 2.3.1). Formálně tedy můžeme zapsat: $L0 \supseteq L1 \supseteq L2 \supseteq L3$.

Třída 0

gramatiky typu 0 - na přepisovací pravidla těchto gramatik nejsou kladena žádná omezení. Někdy jsou nezývány také jako gramatiky bez omezení, nebo fázové gramatiky. Generují rekurzivně spočetné jazyky L0, které mohou být rozpoznány vhodným Turingovým strojem.

Příklad gramatiky typu 0:

G = ({A,B},{a,b},P,A) s pravidly
A → AbB | a
AbB → baB | BAbB
B → b| ε

Třída 1

gramatiky typu 1 (kontextové) - generují kontextové jazyky L1. Tyto gramatiky se skládají z pravidel $\boldsymbol{\alpha A \beta \rightarrow \alpha \gamma \beta}$, kde A je neterminál a α, β a γ řetězce terminálů a neterminálů. Řetězce α a β mohou být prázdné, ale γ musí být neprázdná. Jedinou výjimku tvoří pravidlo $\mathbf{S \rightarrow \varepsilon}$, pokud se S nevyskytuje na pravé straně žádného pravidla. Tyto jazyky jsou rozpoznatelné lineárně ohraničeným Turingovým strojem.

Příklad kontextové gramatiky:

G = ({A,S},{0,1,c},P,S) s pravidly
S → 0A1
0A → 00A1 (α=0,β=e,γ=0A1)
A → c

Třída 2

gramatiky typu 2 (bezkontextové gramatiky) - generují bezkontextové jazyky L2. Skládají se z pravidel $\mathbf{A \rightarrow \gamma}$ s neterminálem A a řetězcem terminálů a neterminálů γ. Pravidlo $\mathbf{S \rightarrow \varepsilon}$ je opět povoleno, pokud se *S* nevyskytuje na pravé straně žádného pravidla. Tyto jazyky jsou rozpoznatelné nedeterministickým zásobníkovým automatem.

Příklad bezkontextové gramatiky:

G = ({S},{0,1,c},P,S) s pravidly
S → 0S1|c

Třída 3

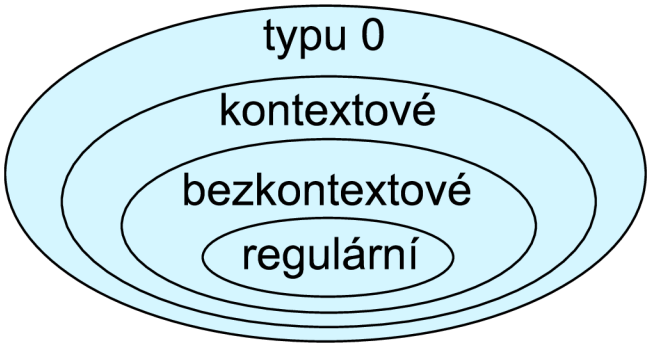
Gramatiky typu 3 (regulární gramatiky) generují regulární jazyky L3. Pravidla těchto gramatik obsahují jeden neterminál na levé straně a řetězec terminálů, který může být následován jedním neterminálem, na straně pravé. Stejně jako u předchozích dvou gramatik tvoří výjimku pravidlo **S** → **ε** , pokud se S nevyskytuje na pravé straně žádného pravidla. Tyto jazyky jsou právě jazyky rozpoznatelné konečným automatem.

Příklad regulární gramatiky:

G = ({A,B},{a,b,c},P,A) s pravidly

A → aaB|ccB

B → bB|ε



Obr. 2.3.1 hierarchičnost jazyků

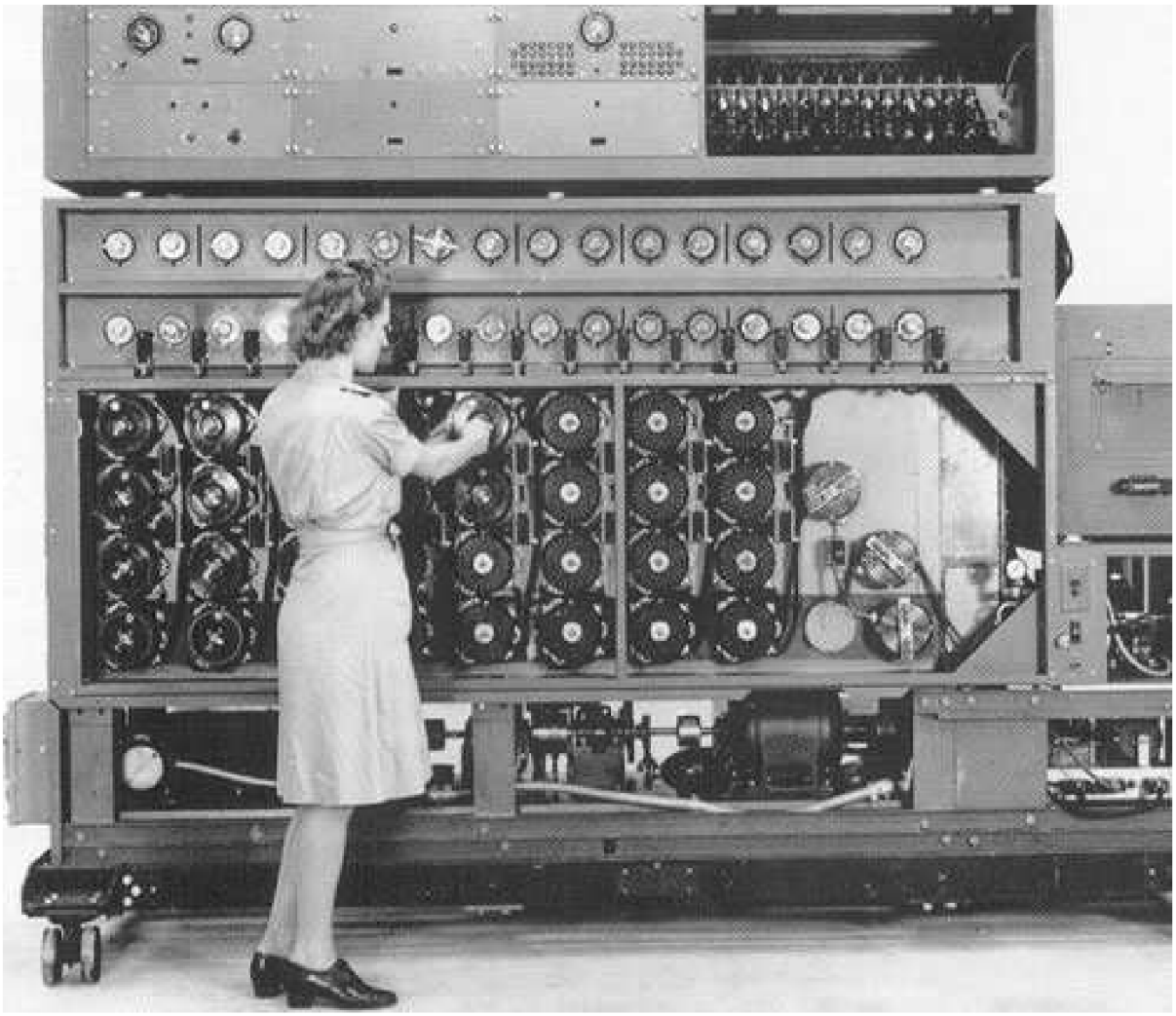
3 Automaty

Teorie automatů je věda, která vznikla počátkem druhé čtvrtiny dvacátého století. U jejího zrodu stál především britský matematik, logik a kryptograf Alan Turing (1912-1954). V článku „On computable numbers, with an application to the Entscheidungsproblem“, jenž Turing publikoval již v roce 1937, prezentuje svůj hypotetický stroj (Turingův stroj). Jednalo se o model abstraktního stroje, který měl být schopen na základě logiky odpovědět na jakoukoli otázku a měl by být schopný zvládnout Hilbertův problém, tedy rozhodnout o pravdivosti jakéhokoli matematického tvrzení. Právě Turingův stroj nakonec dokázal, že není možné mechanickým strojem rozhodnout pravdivost nebo nepravdivost libovolného matematického výroku.

Turingův stroj byl však příliš složitý pro řešení lehčích úkolů a tak byly zavedeny jednodušší modely. Mezi ně patří také konečný a zásobníkový automat. S těmito modely se běžně setkáváme a to nejen v informatice. Na principu modelu konečného automatu je založena například i světelná signalizace na křižovatkách, automat na kávu, výtah, automatické otevírání dveří a podobně. V informatice se pak uplatňuje pro lexikální analýzu u překladačů a vyhledávání vzorků v textu. Toto vyhledávání se používá jak v různých textových editorech, tak například i ve vyhledávačích na internetu (google, seznam). Zásobníkový automat se pak používá především pro syntaktickou analýzu v překladačích.

Tyto automaty můžeme rozdělit do skupin podle následujících kritérií. Automat může být deterministický, nebo nedeterministický, což bude podrobněji vysvětleno v následujícím textu. Dále je můžeme rozdělit na automaty s výstupem a bez výstupu. Ve zbytku této práce budu uvažovat automaty bez výstupu, tedy akceptory. Jedná se o automaty, které slouží k rozpoznávání, zda vstupní (rozpoznávaný) řetězec patří do jazyka přijímaného automatem či nikoli.

V dalším textu uvedu popis jednotlivých modelů. U každého je uvedena jeho definice, popis činnosti, konfigurace a jednoduchý příklad. Konfigurace nám udává aktuální nastavení automatu, na kterém závisí další krok. Speciální konfigurací je tzv. počáteční konfigurace, ve které se automat nachází při jeho spuštění. U příkladů vždy uvádím jednoduchý popis automatu popř. jazyka, který přijímá, a přikládám jeho grafickou reprezentaci pomocí stavového diagramu. Ten je vyňat z výstupu mnou implementovaného programu, jenž je zadáním této práce. Jen podotknu, že orientace hran není určena klasickými šipkami, nýbrž tloušťkou hran a to ve směru od tlusté k tenké (podrobně viz kapitola 4.5).



Obr 3.1. Jeden z prvních Turingových strojů

3.1 Konečný automat

Definice 3.1.1

Nedeterministický konečný automat (KA) je pětice **M = (Q, Σ, r, s, F)**, kde:

- **Q** je konečná neprázdná 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** je počáteční stav, kde $s \in Q$
- **F** je množina koncových stavů, kde $F \subseteq Q$

Popis činnosti automatu:

Konečný automat je nejjednodušší model založený na konečné množině stavů a pravidel. Na začátku se nachází v počátečním stavu a postupně čte vstupní řetězec zleva doprava po jednotlivých symbolech. Podle právě přečteného symbolu a stavu, ve kterém se aktuálně nachází, se pomocí pravidel rozhoduje do jakého stavu přejít. Vstupní řetězec přijme, pokud se automat po jeho přečtení nachází v koncovém stavu.

Konfigurace určuje aktuální nastavení automatu. Pro KA je to dvojice (q, w), kde $q \in Q$ a $w \in \Sigma^*$. Tedy slovně můžeme říct, že se automat nachází ve stavu q a řetězec w je doposud nepřečtená část vstupního řetězce. Pokud by byl stav q stavem počátečním, řekneme, že se jedná o počáteční konfiguraci.

Nedeterminismus tohoto automatu spočívá v tom, že automat může z jedné konfigurace přejít do více stavů a jeden vstupní řetězec může být tudíž interpretován více způsoby. V některé literatuře se uvádí, že si automat vybírá cestu náhodně, jinde zas že přechází do všech možných stavů (větví se a pokud aspoň jedna větev po přečtení vstupu skončí v koncovém stavu, vstup je přijat). V praxi je tento automat ale v podstatě nepoužitelný. Pro jeden vstup potřebujeme pevně určit, zda do daného jazyka patří, či nikoli. Proto je výhodné se nedeterminismu zbavit, přičemž platí, že každý nedeterministický automat se dá převést na automat deterministický. Toho lze dosáhnout odstraněním ε-přechodů a následným algoritmem pro odstranění nedeterminismu (založen na slučování a přidávání stavů).

Deterministický KA můžeme dále upravovat a získat tak typy KA, které mohou být v určitých situacích užitečné. Například úplný KA, který se nemůže zaseknout. Tento automat používá principu „pasti“ což je vlastně jeden stav, který je zacyklený pro všechny symboly nad vstupní abecedou. Do tohoto stavu pak vedou všechny nechtěné přechody. Tento automat můžeme ještě dále upravit na dobře specifikovaný KA a minimální KA.

Možnosti reprezentace

Pro reprezentaci KA existují kromě výčtu prvků i jiné možnosti. Nejčastěji používanými způsoby reprezentace KA jsou:

Tabulka přechodové funkce

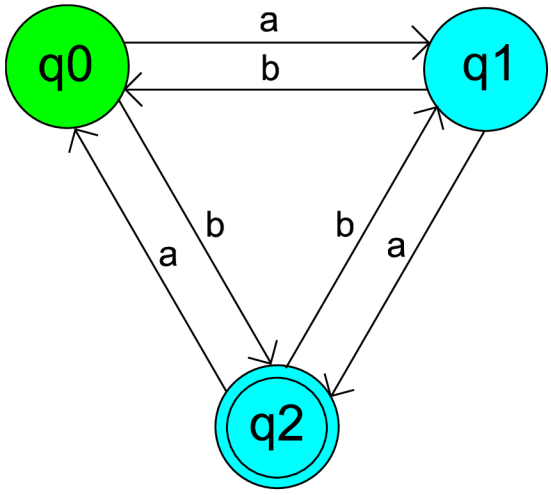
Stavy jsou vypsány v záhlaví řádků, znaky ze vstupní abecedy pak v záhlaví sloupců. Přechodové pravidla jsou určena obsahem vnitřních polí tabulky (tedy obsahují cílové stavy přechodu, který vede ze stavu určeného řádkem po přečtení znaku určeného sloupcem). Počáteční stav bývá označen šipkou a stavy koncové například podtržením.

		a	b
→	q0	q1	q2
	q1	q2	q0
	q2	q0	q1

Stavový diagram

Nejpřehlednější a také nejpoužívanější způsob reprezentace KA. Jedná se o graf, ve kterém uzly znázorňují stavy a ohodnocené orientované hrany přechodové funkce. Znaky, kterými jsou hrany ohodnoceny, pak tvoří vstupní abecedu. Koncové stavy jsou znázorněny dvojitou čarou obvodu stavu.

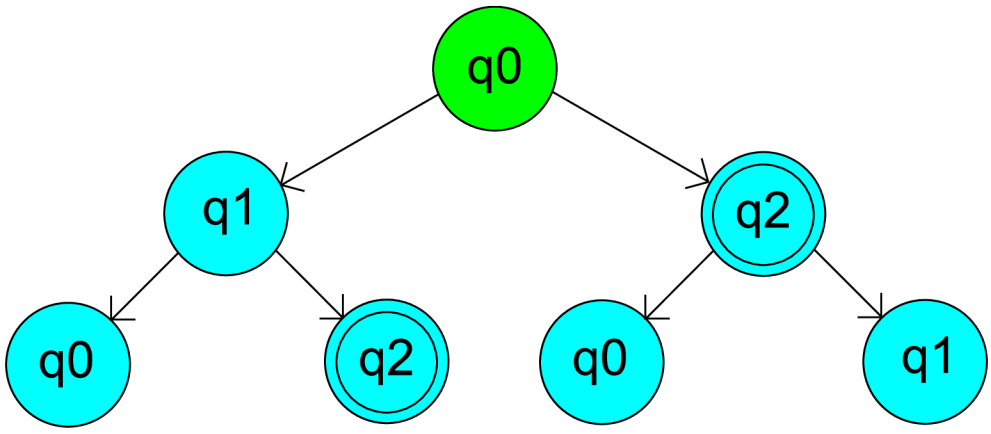
V aplikaci, která je výstupem této práce, používám právě stavové diagramy pro reprezentaci automatů. Všechny tři typy automatů, tedy KA, ZA i TS, lze tímto způsobem znázornit, přičemž se grafy liší pouze v ohodnocení hran (různé formáty pravidel).



Obr. 3.1.1 Stavový diagram.

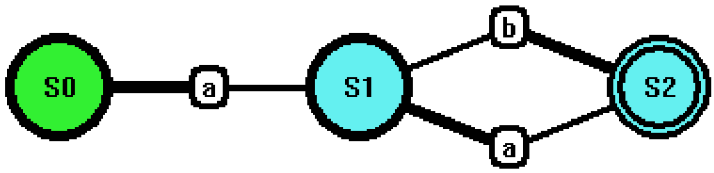
Stavový strom

Pro úplnost zmíním i tuto variantu, i když se příliš často nepoužívá. Kořen stromu zde odpovídá počátečnímu stavu. Z každého uzlu, jenž není listem stromu, vychází tolik hran, kolik má odpovídající stav následníků. Pokud je stav reprezentován více uzly, pak hrany vycházejí pouze z prvního uzlu.



Obr. 3.1.2 Stavový strom.

Příklad:
Jako příklad jsem vybral velice jednoduchý deterministický konečný automat, který má počáteční stav S0 a jeden koncový stav S2. Vstupní abeceda je Σ={a,b} a přijímá jazyk L={aa}.{ba}*.



Obr. 3.1.3 Příklad konečného automatu.

3.2 Zásobníkový automat

Definice 3.1.1
Zásobníkový automat (ZA) je sedmice **M = (Q, Σ, Γ, r, s, S, F)**, kde:

- **Q** je konečná neprázdná množina stavů
- **Σ** je vstupní abeceda
- **Γ** je zásobníková abeceda
- **r** je konečná množina pravidel tvaru: $paA \rightarrow qw$, kde $p, q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $A \in \Gamma$, $w \in \Gamma^*$
- **s** je počáteční stav, kde $s \in Q$
- **S** je počáteční symbol na zásobníku, kde $S \in \Gamma$
- **F** je množina koncových stavů, kde $F \subseteq Q$

Popis činnosti automatu:
Zásobníkový automat se odlišuje od konečného tím, že navíc disponuje neomezenou, zásobníkovou pamětí, typu LIFO (z anglického Last In – First Out). Pracuje stejně jako konečný automat, tedy na začátku se nachází v počátečním stavu, navíc má ještě na zásobníku počáteční symbol. Každý krok znamená, že čtecí hlava přečte jeden symbol ze vstupního řetězce a podle něj, stavu, ve kterém se aktuálně automat nachází, a symbolu, jenž je na vrcholu zásobníku, provede přechod. Než tak provede, odstraní symbol z vrcholu, nebo ho přepíše řetězcem zásobníkových symbolů, které určuje dané pravidlo přechodu. Můžeme stanovit tři druhy kritérií pro přijetí vstupního řetězce.

Daný vstup je přijat, pokud po jeho přečtení:

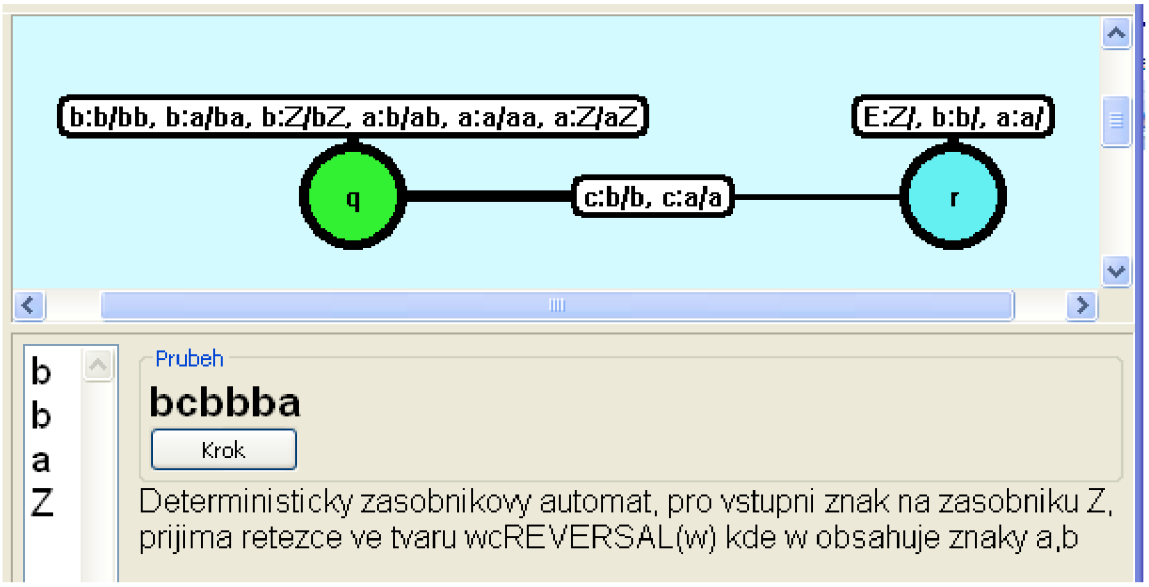
- se automat nachází v koncovém stavu.
- je zásobník prázdný.(nejčastější))
- se automat nachází v koncovém stavu a zásobník je prázdný.

Konfigurace u ZA je trojice (q, w, a), kde $q \in Q$, $w \in \Sigma^*$, $a \in \Gamma$. Konfigurace nám říká, že se automat nachází ve stavu q, řetězec w je doposud nepřečtená část vstupní řetězce a na vrcholu zásobníku je znak a.

Na rozdíl od KA má u ZA nedeterministická verze vyšší výpočetní (rozpoznávací) sílu než deterministický ZA. Definice bezkontextového jazyka nám říká, že jazyk je bezkontextový, právě pokud může být rozpoznán vhodným zásobníkovým automatem. V této definici je ale zásobníkovým

automatem myšlen právě automat nedeterministický. Deterministický pak může rozpoznávat pouze určitou podmnožinu bezkontextových jazyků (tedy deterministické bezkontextové jazyky). Jako ukázkový příklad bezkontextového jazyka, jenž nejde rozpoznat deterministickým ZA, se často uvádí jazyk $\{w(w)^R | w \in \{a, b\}^*\}$. Slovně můžeme říct, že se jedná o jazyk skládající se z konkatenace řetězce nad abecedou jazyka a řetězce tvořeného jeho reversací. Problém spočívá v tom, že deterministický automat nedokáže určit střed vstupního řetězce a tedy neví, kdy přestat na zásobník ukládat symboly a začít je odstraňovat. Pokud mezi tyto dva podřetězce vložíme znak, dokáže tento jazyk rozpoznat i automat deterministický (viz následující příklad).

Příklad:
Tento deterministický zásobníkový automat přijímá jazyk L=w.{c}.*reversal*(w), kde w obsahuje znaky a, b. Na obrázku je automat zachycen v konfigurace (q, "bcbbba", b). Vstupním řetězcem pak byl řetězec "`abbbcbbba`" (automat tedy provedl již tři kroky výpočtu).



Obr. 3.2.1 Příklad zásobníkového automatu

3.3 Turingův stroj

Definice 3.3.1
Turingův stroj (TS) je šestice **M = (Q, Γ, r, b, s, F)**, kde:

- **Q** je konečná neprázdná množina stavů
- **Γ** je konečná abeceda znaků
- **r** je konečná množina pravidel tvaru $pA \rightarrow qB\{L,R,0\}$, kde $p, q \in Q$, $A,B \in \Gamma$ a L,P,0 reprezentují posun (doleva, doprava, bez posunu)
- **b** je symbol reprezentující prázdný symbol, $b \in \Gamma$
- **s** je počáteční stav, kde $s \in Q$
- **F** je množina koncových stavů, kde $F \subseteq Q$

Popis činnosti:
Na začátku se na (teoreticky nekonečnou) pásku zapíše slovo a nastaví se čtecí/zapisovací (r/w) hlava na určitou pozici. Slovo je ohraničeno prázdnými znaky (v dalším textu budu používat jako prázdný znak #). Podle přečteného znaku a stavu, ve kterém se automat nachází, se řídící jednotka rozhodne, jaké písmeno na pásku zapíše, do jakého stavu přejde a zda se hlava posune o znak doleva, doprava, nebo zůstane na místě. Práce Turingova stroje je ukončena, pokud nelze použít žádnou další instrukci.

Konfigurace pro TS sestává z trojice (q, w, n), kde $q \in Q$, $w \in \Gamma^*$ a $n \in N$. Konfigurace nám říká, že se stroj nachází ve stavu q, řetězec w je páska a číslo n udává pozici r/w hlavy na pásce.

Z popisu činnosti lze odtušit, že oproti KA se TS liší ve dvou základních věcech. První rozdíl spočívá v pásce, na níž je na začátku výpočtu zapsáno slovo a která je (teoreticky) oboustranně nekonečná (někdy jen zprava nekonečná). Hlava spojená s řídící jednotkou se pak může po této pásce pohybovat oběma směry. Druhý rozdíl je ten, že hlava může symboly nejenom číst, ale i zapisovat.

TS můžeme rozdělit podle jejich použití do tří základních kategorií:

TS řešící problémy
Pomocí tohoto typu TS můžeme řešit nějaký problém P, což spočívá v tom, že ke každému vstupu stroj M provede výpočet pomocí předepsaných instrukcí a řešení uloží na výstup.

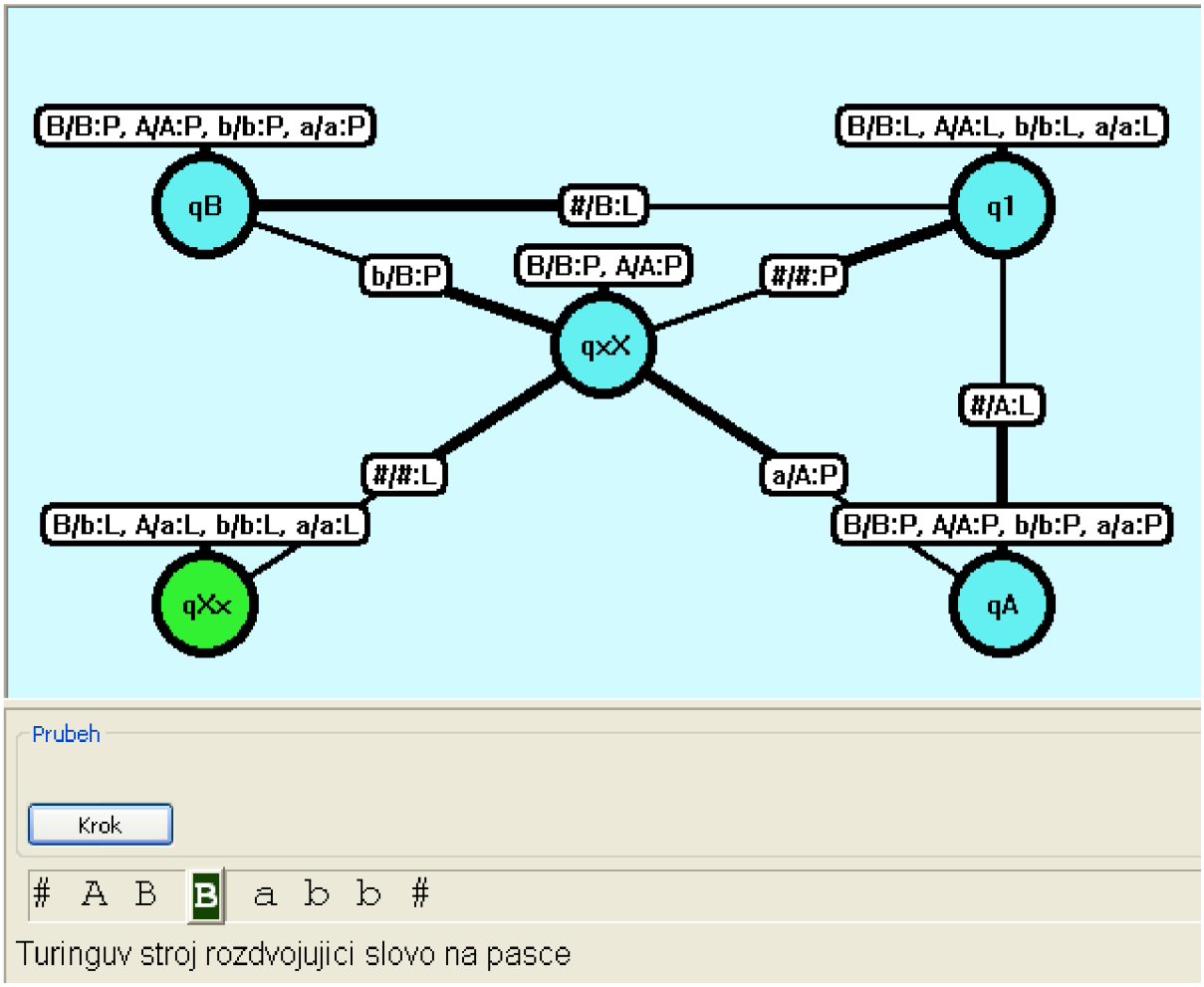
TS rozhodující ANO/NE
Jsou založeny na dvou koncových stavech *accept* (přijmout) a *reject* (zamítnout). Pomocí těchto strojů můžeme rozhodovat nejrůznější boolovské výrazy.

TS rozpoznávající jazyky
Zde platí, že každý jazyk typu 0 je přijímán nějakým TS a to tak že (definice 6.2.1 z opory TIN[2]):

1. Řetězec $w \in \Sigma^*$ je přijat TS $M = (Q,\Sigma, \Gamma, \delta, q0, qF)$, jestliže M při aktivaci z počáteční konfigurace pásky #w#... a počátečního stavu q0 zastaví přechodem do koncového stavu qF , tj. $(q0,\#w\#\omega, 0)^* \vdash M(qF , \gamma, n)$ pro nějaké $\gamma \in \Gamma^*$ a $n \in N$.
2. Množinu $L(M) = \{w \mid w \text{ je přijat TS } M\} \subseteq \Sigma^*$ nazýváme jazyk přijímaný TS M.

Dále rozeznáváme TS jednopáskové a vícepáskové, přičemž platí, že k libovolnému n-páskovému TS existuje TS jednopáskový. Stejně je tomu i u nedeterminismu, tedy pro každý NTS existuje DST. Z toho vyplývá, že zvýšením paměťových možností, či zavedením nedeterminismu se u TS nezvyšuje jeho výpočetní síla (schopnost přijímat jazyky).

Příklad:
Jako příklad TS jsem zvolil již trošku složitější model než v předchozích případech. Jedná se o model, který rozdvojí řetězec nad abecedou {a, b}. Například pro aktuální konfiguraci na obrázku platí, že se automat nachází ve stavu qXx a r/w hlava ukazuje na pozici pásky se symbolem `'B'`. Nad touto konfigurací můžeme použít pouze pravidlo zapsané jako B/b:L. Při dalším kroku by pak podle tohoto pravidla r/w hlava přepsala aktuální symbol `'B'` na symbol `'b'` a posunula se o jednu pozici doleva, přičemž by automat zůstal ve stejném stavu.



Obr 3.3.1Příklad Turingova stroje

4 Návrh implementace

4.1 Specifikace požadavků

Zadání praktické části této práce je vytvoření programu pro vizualizaci práce konečného automatu, zásobníkového automatu a Turingova stroje. Tato aplikace by měla sloužit pro podporu výuky a to především pro demonstraci práce automatů například na přednáškách a podobně. Zároveň by ale měla být natolik jednoduchá na ovládání, aby poskytovala i studentům možnost navrhování vlastních modelů a následné experimentování.

Pro začátek je vhodné stanovit si základní požadavky, které by měla aplikace splňovat:

- Intuitivní uživatelské prostředí
- Načítání automatu přes externí soubor
- Co nejjednodušší syntaxi vstupního souboru pro popis automatu
- Zadávání vlastního vstupního řetězce v aplikaci
- Možnost krokování práce automatu

Dále jsem se rozhodl neklást omezení na přechodová pravidla, čímž je dosaženo možnosti implementace jak deterministických, tak i nedeterministických modelů.

4.2 Výběr vývojového prostředí

Jako vývojové prostředí jsem si zvolil objektově orientovaný jazyk C++ u kterého jsem využil jeho zpětnou kompatibilitu s procedurálním jazykem C pro implementaci jádra programu. Pro tvorbu uživatelského prostředí pak využívám volně šiřitelné GUI knihovny wxWidgets. Velkou předností této knihovny je, že je multiplatformní, což znamená, že programy napsané pomocí těchto knihoven jsou přenositelné mezi jednotlivými operačními systémy (Windows, Linux, MacOS), aniž by se muselo zasahovat (nebo jen lehce zasahovat) do jejich zdrojového kódu.

Aplikaci jsem psal v programu Code::Blocks verze 0.82 na platformě MS Windows XP, kde byl také zkompilován. Použitá verze knihovny wxWidgets 2.8.9.

4.3 Syntaxe jazyka pro popis automatu

Jedním z prvních úkolů bylo navrhnout syntaxi jednoduchého jazyka pro popis automatů. Ze vstupního souboru, popisujícího automat, je nutné zjistit typ automatu a data, kterými je daný automat definován. Pro zjednodušení jsem se rozhodl vypustit vstupní abecedu. Ta sice definuje daný automat, ale dá se vyčíst z pravidel přechodů.

Syntaxe vstupního souboru:

Je nutné dodržovat pořadí příkazů. Ty musí být odděleny středníkem, přičemž mezi jednotlivými lexémy může být neomezený počet bílých znaků (mezera, tabulátor,…).

Na začátku souboru se určí typ automatu jedním z klíčových slov `FA`, `PDA`, nebo `TM` (finite automata, pushdown automata, Turing machine). U zásobníkového pak následuje počáteční zásobníkový symbol v kulatých závorkách.

V definování vlastního automatu se začíná výčtem stavů a jejich souřadnic. Tento příkaz začíná klíčovým slovem `states` za nímž následuje výčet stavů ve složených závorkách. Jednotlivé stavy jsou ve formátu název[pozice_x, pozice_y] a oddělují se čárkou. V tomto příkazu také nastavujeme vlastnost jednotlivým stavům. První ve výčtu je automaticky stavem počátečním a každý stav, který označíme znakem mřížky je koncový.

Příklad použití příkazu:

```
states {S1[50,50], S2[50,100], #S3[200,200]};
```

Tento automat bude obsahovat tři stavy S1, S2 a S3, z nichž S1 bude stavem počátečním a S3 koncovým.

Dále může následovat nepovinný textový popis automatu ve formátu: `descr "text".` Nyní už zbývá jen nadefinovat jednotlivé pravidla přechodů (instrukce). V těch se sice každý automat liší, ale některé prvky jsou stejné, čehož se dá využít pro zjednodušení načítání. Každé pravidlo obsahuje dvě části, tedy konfiguraci, z které má automat přejít, a změny, které má provést. Tyto části se oddělují rovnítkem. Pokud potřebujeme zadat prázdný symbol, značíme `''`, pokud prázdný řetězec, pak `""`. Příklady pravidel daných automatů:

Konečný automat:

```
(S1, 'a') = (S3);
```

Pokud se automat nachází ve stavu S1 a na vstup přijde symbol 'a', pak automat přejde do stavu S3.

Zásobníkový automat:

```
(S1 ,'', 'Z') = (S2, "aZ");
```

Pokud se automat nachází ve stavu S1, na vstupu je jakýkoliv symbol a na vrcholu zásobníku je symbol Z, pak automat přejde do stavu S2 a na vrchol zásobníku se zapíše symbol 'a', přičemž symbol ze vstupu nebude odebrán (přečten).

Turingův stroj:

```
(S1 ,'a') = (S2 ,'A', P);
```

Pokud se automat nachází ve stavu S1 a čtecí hlava přečte na pásce znak 'a', pak automat přejde do stavu S2, přepíše původní znak 'a' na znak 'A' a posune se na pásce o jedno políčko doprava.

Příklad textového souboru popisujícího zásobníkový automat:

```
PDA(Z);

states{p[200,200], q[500,200]};

descr "Deterministicky zasobnikovy automat s pocatecnim
zasobnikovym znakem Z, prijima retezce ve tvaru wcREVERSAL(w)
kde w obsahuje znaky z abecedy {a,b}";

(p ,'a', 'Z') = (p, "aZ");
(p ,'a', 'a') = (p, "aa");
(p ,'a', 'b') = (p, "ab");
(p ,'b', 'Z') = (p, "bZ");
(p ,'b', 'a') = (p, "ba");
(p ,'b', 'b') = (p, "bb");
(p ,'c', 'a') = (q, "a");
(p ,'c', 'b') = (q, "b");
(q ,'a', 'a') = (q, "");
(q ,'b', 'b') = (q, "");
(q ,'', 'Z') = (q, "");
```

4.4 Načtení automatu

Když už máme nadefinovanou syntaxi pro vstup, musíme vyřešit jeho načtení programem a formát, do kterého si automat načte. Pro rozlišování jednotlivých lexémů ze vstupního souboru jsem vytvořil funkci `nactiToken`, což je jednoduchý lexikální analyzátor, který pracuje právě na principu deterministického konečného automatu. V cyklu provádí jednotlivé kroky:

1. načte znak ze vstupního souboru
2. pokud může, provede přechod s tímto znakem
   pokud ne, daný lexém není přijat a program zahlásí chybu
3. pokud se program nachází v koncovém stavu, funkce vrátí token a jeho typ
   pokud ne, vrátí se ke kroku 1.

Tuto funkci pak volá funkce `nactiAutomat`, která implementuje velice jednoduchý syntaktický analyzátor. Volá si postupně tokeny, rozhoduje, zda na dané místo patří, a ukládá informace o automatu.

Jelikož předem nevíme, kolik bude automat obsahovat stavů, pravidel a přechodů, rozhodl jsem se ho ukládat jako dynamickou datovou strukturu tvořenou dynamickými lineárními seznamy. Pro uložení informací slouží čtyři struktury: *tautomat, tstav, tprechod a tpravidlo*. Ty obsahují jednak ukazatele pro zřetězení s ostatními entitami, jednak data vztahující se k dané struktuře. Pro jednotlivé struktury jsou to:

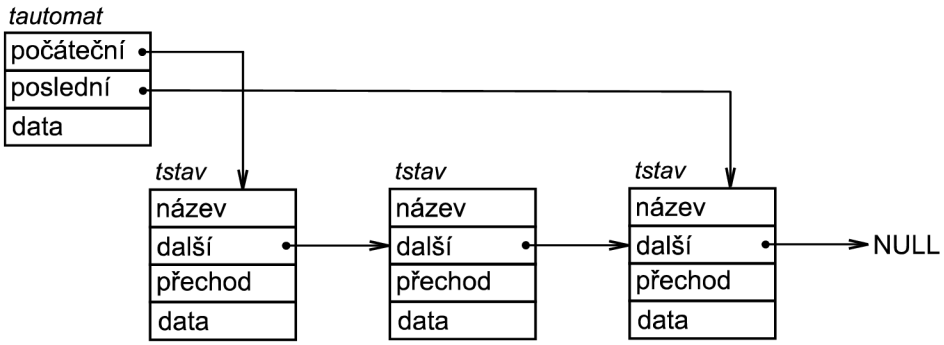
tautomat: typ automatu, textový popis a v případě, že se jedná o zásobníkový automat, také počáteční zásobníkový symbol.

tstav: informace o tom, zda je stav koncový a o jeho pozici.

tprechod: počet pravidel.

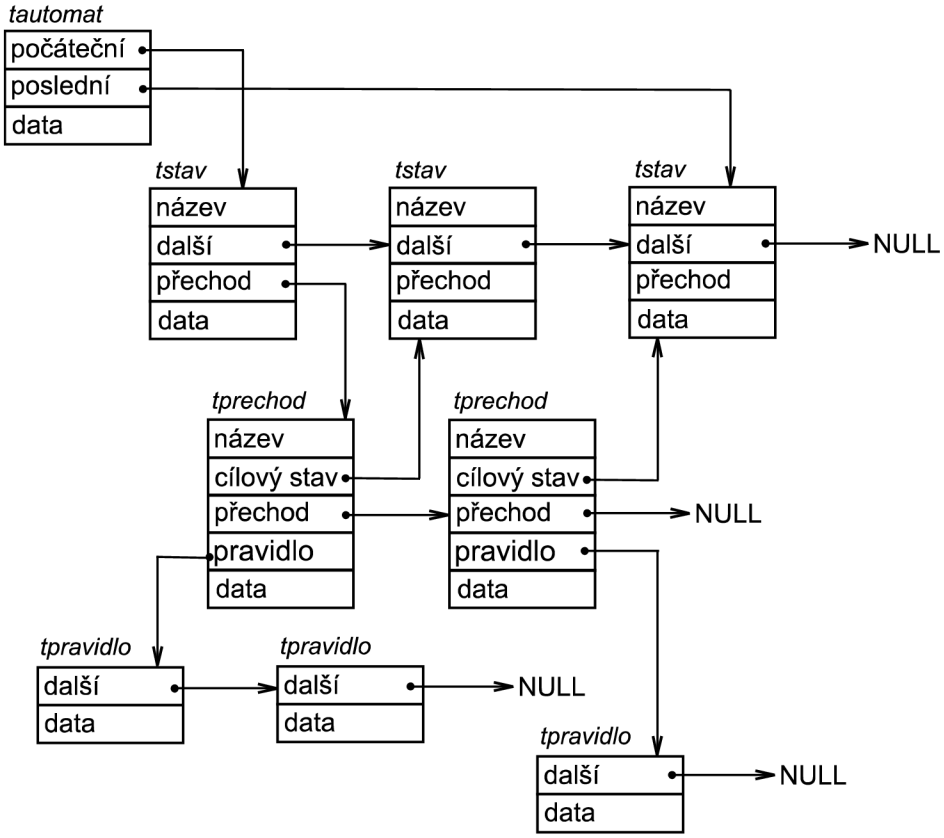
tpravidlo: podle typu automatu jsou to: znak na vstupu, na zásobníku (popř. na pásce u Turinga), řetězec pro přepsání zásobníkového symbolu, znak pro přepsání znaku na pásce a posun pásky.

Nejprve se načtou stavy do lineárního dynamického seznamu (viz. Obr. 4.2.1).



Obr. 4.2.1.Lineární seznam pro načtení stavů

Poté se načítají pravidla přechodů. Během implementace programu jsem zjistil, že nejvýhodnější pro příští práci s automatem a především jeho grafickou reprezentaci je vytvoření pomocné struktury přechodů, ke kterým se budou pravidla vztahovat. To znamená, že se u každého pravidla, které bude ze vstupu přečteno, zjistí, zda už neexistuje přechod se stejným cílovým stavem. Pokud ano, pravidlo bude zařazeno k tomuto přechodu a pokud ne, vytvoří se přechod nový. Schéma celé dynamické struktury automatu je na následujícím obrázku.



Obr. 4.2.2.schéma dynamické struktury, ve které je automat načten

4.5 Uživatelské rozhraní

Uživatelské rozhraní by mělo být intuitivní a co možná nejjednodušší. Navrhl jsem tedy GUI podle klasického konceptu, který je uživateli dobře znám a umožňuje mu rychlé ztotožnění s ovládáním aplikace. Nejprve je nutné si stanovit základní prvky, které jsou nepostradatelné pro chod a ovládání aplikace. Jsou to:

- Načtení automatu
- Plátno s grafickou reprezentací automatu
- Pole pro zadání vstupního řetězce
- Spuštění simulace
- Krokování simulace a přehled jeho průběhu

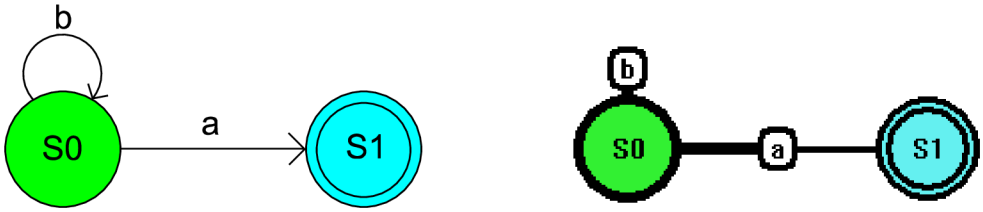
Při načítání jednotlivých typů automatu musíme brát ohled na specifikace jejich popisu. U konečného a zásobníkového automatu potřebujeme zobrazovat nenačtenou část vstupního řetězce v aktuálním kroku. U zásobníkového navíc jeho paměť, tedy zásobník. U Turingova stroje je vstup zapsán na pásku, na které musí být zviditelněna aktuální pozice čtecí/zapisovací hlavy. Navíc, jelikož můžeme pracovat i s nedeterministickými automaty, je zapotřebí zobrazovat tabulku možných přechodů a jakmile může automat přejít z dané konfigurace do dvou a více různých konfigurací, musí mít uživatel možnost zvolit si pravidlo pro přechod.

Otevření souboru je začleněno do horního menu baru, z kterého jde dále zjistit informace o programu, nebo celou aplikaci zavřít. Hlavní okno aplikace je rozděleno na tři části. Vlevo se nachází panel pro spuštění simulace, která obsahuje pole pro zadání vstupního řetězce a tlačítko pro spuštění simulace automatu. Dále tento panel obsahuje informaci o vstupním zásobníkovém znaku, pokud je načten zásobníkový automat, čtyř-tlačítkový ovládací prvek pro nastavování velikosti plátna a tabulku zobrazující možná pravidla přechodů. Pravá část obrazovky je rozdělena na dvě části: plátno pro vykreslování stavového diagramu automatu a panel s průběhem simulace. Tento panel se mění podle typu načteného automatu tak, aby neobsahoval nepotřebné prvky.

4.6 Grafická reprezentace

Pro kreslení se ve vxWidgets používá tzv. *device kontext* (dále jen DC). Neexistuje zde možnost kreslit přímo do okna. Místo toho se pro okno vytvoří DC, do kterého se následně kreslí. Existuje řada tříd DC, já jsem si zvolil klasickou třídu `wxPaintDC`, kde však docházelo k nepříjemnému blikání grafu při krokování. Tento problém nakonec vyřešil přechod na `wxBufferedDC`, jenž pracuje pomocí dvojitého bufferu.

S implementací grafických prvků v aplikaci jsem neměl skoro žádné zkušenosti a tak pro mě byla tato část nejobtížnější. Po nějaké době experimentování s grafickými nástroji jsem se rozhodl pro vlastní úpravu stavového diagramu. Snahou bylo zjednodušení diagramu při zachování co nejvyšší přehlednosti. Úpravy se týkají zobrazení přechodů. Jednak jde o znaky přechodu, které nejsou posazené nad hranou, nýbrž v buňce uprostřed hrany. Druhou změnou je již zmiňované znázornění orientace hran, které není specifikováno klasicky šipkou, ale tloušťkou hrany. Ta je silnější od stavu, ze kterého přechod vychází až do středu hrany a slabší odsud po stav cílový (viz Obr 4.5.1).



Obr 4.5.1 znázorňuje způsob zobrazení klasickým stavovým diagramem a upraveným diagramem, který jsem použil v aplikaci.

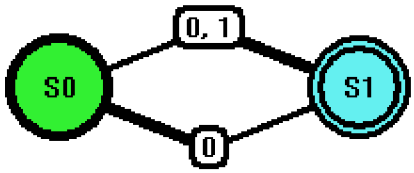
Pravidla pro přechod jsou znázorněna v grafu v buňce (uprostřed hran) a jsou oddělena čárkou. Pro znázornění prázdného znaku nebo řetězce je použit symbol E. Syntaxe se pak liší podle typu automatu:

Konečný automat
Pouze symbol ze vstupní abecedy jako podmínka provedení přechodu při načtení symbolu ze vstupního řetězce.

Zásobníkový automat
Řetězec ve formátu a:X/w, kde a značí symbol ze vstupního řetězce, X symbol nacházející se na vrcholu zásobníku a w řetězec, jenž je posléze na zásobník zapsán.

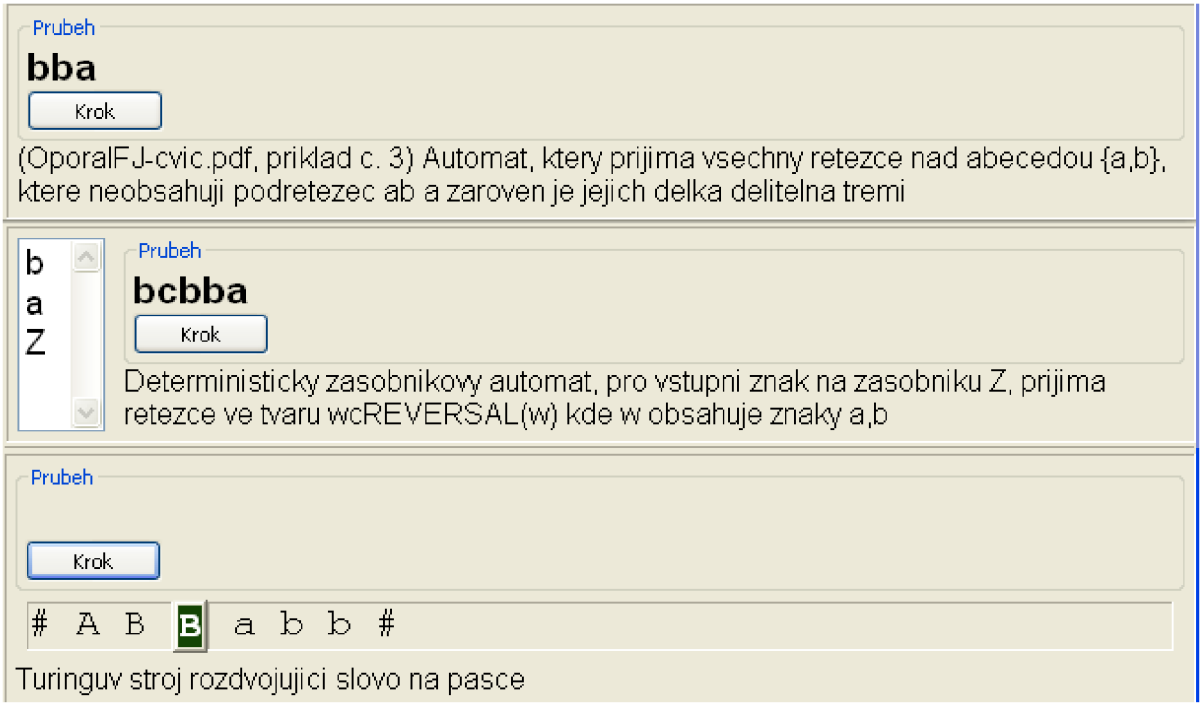
Turingův stroj
Řetězec ve formátu A/B:P, kde A značí symbol načtený z pásky, B symbol, jenž je na pásku zapsán a P vyjadřuje posun hlavy na pásce po provedení zápisu (L pro posun doleva, P pro posun doprava a symbol – pokud k posunu nedojde).

Koncové stavy jsou znázorněny dvojitou čarou obvodu stavu. Krokování simulace probíhá pomocí tlačítka ve spodním panelu, kde můžeme také sledovat nenačtenou část vstupního řetězce, popřípadě zásobník či pásku u ZA a TS. V grafu je pak krok (pokud je proveditelný) znázorněn změnou aktuálního stavu, tedy barevným rozlišením (aktuální stav má zelenou výplň). Při vykreslování automatu je také zapotřebí zjišťovat, zda se nejedná o obousměrný přechod a pokud ano, vykreslit ho s odchylkou aby nedošlo k překreslení opačným přechodem (viz Obr. 4.5.2).



Obr 4.5.2Vykreslení obousměrného přechodu

Spodní panel s průběhem simulace se mění podle typu načteného automatu tak, aby neobsahoval zbytečné prvky (viz Obr 4.5.3).



Obr 4.5.3 znázorňuje panel pro průběh simulace pro jednotlivé automaty (shora KA, ZA, TS)

Vykreslení automatu pak obstarává funkce `vykresliAutomat`. Provádí výpočet středu hrany a odchylky v případě obousměrného přechodu a postupně vykresluje hrany, buňky a stavy.

5 Testování

Pro testování aplikace a ukázku jejího využití jsem vytvořil sadu dvanácti modelů (pro každý typ automatu čtyři). Demonstrační automaty jsem se snažil vybírat tak, aby ukázaly a otestovaly možnosti aplikace a především její schopnost pracovat s jednotlivými typy a modifikacemi těchto tří automatů. Během testování a experimentování jsem nenarazil na jediný automat, který by program nedokázal interpretovat a odsimulovat. Aplikace dokáže demonstrovat práci jak deterministických, tak i nedeterministických variant všech tří automatů. Konečné automaty úplné, dobře specifikované i minimální. U zásobníkových automatů je aplikace vhodná pro demonstraci vyšší výpočetní síly nedeterministické verze, nebo pro ukázku různých kritérií pro přijetí slova. U Turingových strojů byly testovány všechny verze uváděné v teoretické části práce, tedy TS řešících nějaký problém (s výstupem na pásku), TS rozhodující axiom i TS pro přijímání jazyků. Aplikace bez problémů zvládá i složitější modely s mnoha stavy. Právě tato všestrannost si myslím patří k hlavním kladům programu.

6 Závěr

Zadání této práce bylo splněno. Výsledkem je jednoduchá aplikace, která umožňuje demonstrovat práci konečných automatů, zásobníkových automatů a Turingových strojů. Prvním bodem zadání bylo prostudovat teorii automatů. Touto teorií a s ní související teorií formálních jazyků se zabývá kapitola dvě a tři. Návrhem a řešením implementace vlastního programu pak kapitola čtvrtá, ve které se nachází také návrh rozhraní pro popis automatů (podkapitola 4.3).

Na internetu je dostupná řada programů pro ukázku práce s automaty. Drtivá většina je však zaměřena pouze na jeden typ a málokterý program obsahuje grafickou reprezentaci stavovým diagramem, který je přitom pro názorné zobrazení ideální. Největší výhodu mnou implementované aplikace vidím především v rozsahu typů a variant automatů, které může interpretovat a odsimulovat. Zde nejsou kladena téměř žádná omezení a v tomto směru by snad jediným rozšířením mohla být implementace vícepáskových Turingových strojů. Dále by se dala práce rozšířit o funkci umožňující vrátit zpět alespoň jeden krok simulace. Pro urychlení vytváření automatů by mohl být implementován editor s grafickým uživatelským rozhraním. Tím by se uživatel vyhnul nutnosti prostudovat syntaxi rozhraní pro popis těchto automatů.

Aplikace by mohla najít uplatnění především jako pomůcka pro podporu výuky na přednáškách. Práce s ní je dostatečně jednoduchá a poskytuje tak možnost vytváření modelů automatů a jejich simulaci pro širší okruh uživatelů.

Pro mě osobně bylo největším přínosem při vytváření této práce seznámení se s grafickou knihovnou wxWidgets. Získal jsem tak základní znalosti o tvorbě grafického rozhraní v této knihovně a o nástrojích určených pro vykreslování grafických výstupů.

Literatura

[1] Chytil, M. *Automaty a gramatiky.* Praha: SNTL, 1984.

[2] Černá, I., Křetínský, M., Kučera, A. *Automaty a formální jazyky I.* Brno: Masarykova univerzita. Fakulta informatiky, 2002.
URL: http://is.muni.cz/elportal/estud/fi/js06/ib005/Formalni_jazyky_a_automaty_I.pdf

[2] Meduna ,A., Lukáš, R. *Formální jazyky a překladače.* Opora předmětu IFJ. VUT Brno.

[4] Jančar, P. *Teoretická informatika:* on-line učební text. 1. vyd. Ostrava: Ediční středisko VŠB-TUO, 2007, 330 s. ISBN 978-80-248-1487-2.
URL: http://www.cs.vsb.cz/jancar/TEORET-INF/ti-text.2007-08-31.pdf

[5] Češka M., Vojnar T., Smrčka A. *Teoretická informatika*, VUT Brno.
URL: http://www.fit.vutbr.cz/study/courses/TIN/public/Texty/oporaTIN.pdf

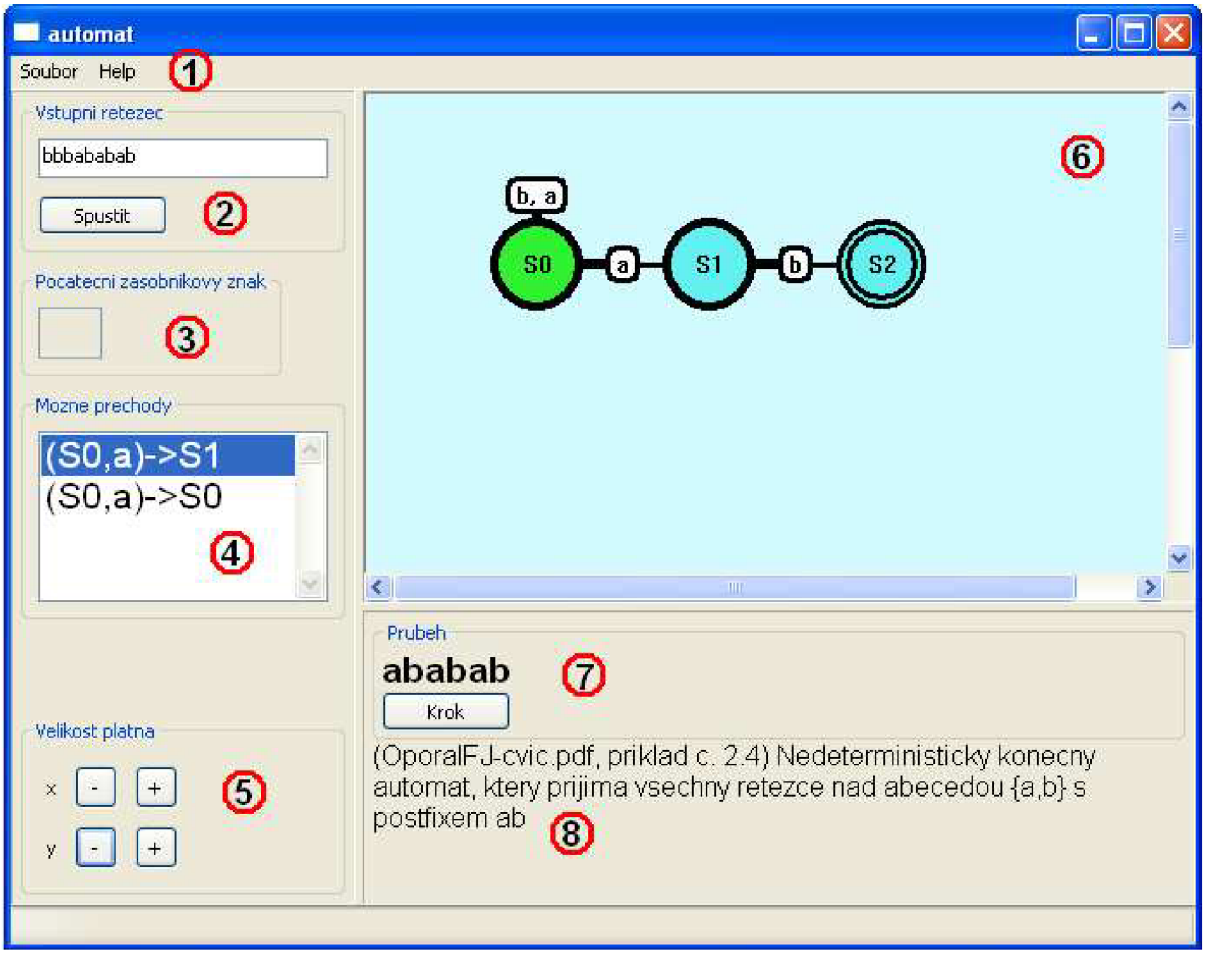
[6] Internetová encyklopedie http://www.wikipedia.org/

[7] Smart, J.; Hock, K.; Csomor, S.: *Cross-Platform GUI Programming with wxWidgets*. Prentice Hall, 2006, ISBN 0-13-147381-6.

Seznam příloh

Příloha 1. Manuál
Příloha 2. CD (obsahuje zdrojové kódy, spustitelnou aplikaci, sadu dvanácti ukázkových automatů a tuto technickou zprávu)

Příloha 1: manuál



1. Menubar obsahuje 3 položky: pro načtení souboru, ukončení aplikace a výpis informací o programu
2. Okno pro zadání vstupního řetězce. Při simulaci Turingova stroje je možné nastavit r/w hlavu na určitou pozici umístěním symbolu '|' před znak který má být prvně přečten.
3. U ZA je zde uveden počáteční zásobníkový znak.
4. Výpis možných přechodů. U nedeterministického automatu možnost volby přechodu.
5. Ovládání velikosti plátna.
6. Plátno, na které se vykresluje automat pomocí stavového diagramu.
7. Panel s údaji o průběhu simulace (doposud nenačtená část vstupního řetězce, zásobník, popř. páska) s tlačítkem pro provedení kroku. Mění se podle typu načteného automatu (viz. Obr. 4.5.3).
8. Nepovinný popis automatu.